# Answering Factoid Questions via Ontologies : A Natural Language Generation Approach

Bikash Gyawali

M.Sc. Dissertation

Department of Intelligent Computer Systems
Faculty of Information and Communication Technology
University of Malta
September 15, 2011

Supervisor: Dr. Albert Gatt

Submitted in partial fulfillment of the requirements for the Degree of European Master of Science in Human Language Science and Technology (HLST)

# Answering Factoid Questions via Ontologies : A Natural Language Generation Approach

Bikash Gyawali

M.Sc. Dissertation


Department of Mathematics and Computer Science
University of Nancy 2
September 15, 2011

Supervisor: Dr. Maxime Amblard

Submitted in partial fulfillment of the requirements for the Degree of European Masters Program in Language and Communication Technologies (LCT)

# M.Sc.(HLST)
# FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
# UNIVERSITY OF MALTA

## Declaration

Plagiarism is defined as "the unacknowledged use, as one's own work, of work of another person, whether or not such work has been published, and as may be further elaborated in Faculty or University guidelines". (University Assessment Regulations, 2009, Regulation 39 (b)(i), University of Malta).

I, the undersigned, declare that the Master's dissertation submitted is my own work, except where acknowledged and referenced.

I understand that the penalties for making a false declaration may include, but are not limited to, loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Bikash Gyawali _____

Student Name | Signature

CSA5310 HLST Dissertation

Course Code

Answering Factoid Questions via Ontologies : A Natural Language Generation Approach

Title of work submitted

Date : September 15, 2011

# Abstract

We present a systematic approach to the generation of natural language descriptions of logical facts from ontologies. We design and discuss our Natural Language Generation (NLG) architecture in terms of implementing a factoid question answering platform upon ontologies; we identify what questions can be asked upon the knowledge base, determine relevant contents from the knowledge base that best serve in generating response to the questions and process those contents confirming to the popular patterns of expression, as identified from a survey, in order to generate answers in natural language (English); all of this while justifying the rationale of the approach and the possible benefits such systems can offer.

# Acknowledgments

I take this opportunity to express my sincere gratitude towards people whose direct and indirect support has helped me to come up with this work. I am highly indebted to my parents for all the pains and efforts they have taken throughout years in bringing me up and guiding me on the glorious path of education. My brother deserves credit for motivating and persuading me in times of difficulties; he has always been my inspirational figure. My direct supervisor, Dr. Albert Gatt at University of Malta, has put immense effort and time upon me to ensure the success of this thesis. He introduced "Natural Language Generation (NLG)" to me and guided me throughout this work. His relentless follow ups, guidance and correction has helped me materialize this work; many thanks indeed. I am grateful for the support of my co-supervisor, Dr. Maxime Amblard at University of Nancy 2, in helping me with suggestions for this work and also in facilitating me with fulfilling the academic and administrative formalities in his additional role of a local program coordinator at University of Nancy 2. Many thanks to Dr. Mike Rosner and Dr. Chris Staff at University of Malta and Prof. Dr. Guy Perrier and Dr. Patrick Blackburn (formerly) at University of Nancy 2, whose support has made my stay in the program fruitful. Finally, I would like to thank the EU for providing generous financial assistance throughout my studies in the Erasmus Mundus European Masters Program in Language and Communication Technologies.

# Contents

# Chapter 1

# Introduction

With the advent of the "Semantic Web Vision", ontologies have become the formalism of choice for knowledge representation and reasoning. Usually, ontologies are authored to represent real world knowledge in terms of concepts, individuals and relations in some variety of Description Logic.

While ontologies are well suited for computational reasoning, it may also be equally less intuitive and less insightful for a human user to grasp all the concepts and relationships that exist in the given ontology, at a glance. This can be true for experts who are working on a broad scale ontology and need to analyze a fairly large chunk of already present logical statements in order to add/modify newer axioms to the knowledge base and also for beginners trying to get acquainted with the ontology formalism. Thus, in the course of design, maintenance and description of ontologies, a human user would perhaps like to seek necessary information from a given ontology in some natural language (like English). In such cases, it would be desirable to have access to a natural language description of the knowledge present within the ontology.

Since ontologies are used to describe real world knowledge in a formally prescribed structure, one can consider an ontology to be an organized knowledge source repository, which under suitable interpretation, could also serve as an input to a class of systems – referred to as Natural Language Generation(NLG) systems in the literature – to generate textual descriptions of data in some natural language like English.

In this Master thesis, we work in developing a system that can help general users in retrieving natural language description (in English) of information present within an ontology as answers to a set of questions posed upon the ontology knowledge base. We discuss and present our NLG architecture; encompassing generic approaches for retrieving, processing and expressing contents from ontologies so as to express them as answers in natural language to the questions posed.

# Chapter 2

# Ontology, Question Answering and Natural Language Generation

## 2.1 Ontology

The term "ontology" has its roots in philosophy and was borrowed into the Artificial Intelligence domain as early as the 1980s. Hobbs & Moore [19] regarded it as a theory of a modeled world (a theory which could provide a systematic way for mapping axioms/facts to some representative element in the universe of discourse) while Gruber [18] defined it as an explicit specification of a conceptualization (meaning that an ontology could serve as a formal model for expressing facts and relationships between facts).

Over the years, research on ontologies has been influenced by several other relevant formalisms like Conceptual Graphs [40] and Frames [29]. The focus has been on identifying how expressively and efficiently an ontology can model real world knowledge and how such knowledge bases can be used suitably and productively for computer based processing. The notion of having a formalism expressive enough to meet real world knowledge representation requirements and tractable enough to provide a decidable representation of a knowledge base for logical inferencing has led to several evolutions of ontology formalism. Recent standards for modeling knowledge in an ontology formalism have mainly adopted the OWL family[1] of description logics for expressing facts.

[1] `http://www.w3.org/TR/owl-features/`

Current ontologies are authored in one of several varieties of OWL based representation, namely OWL FULL, OWL DL and OWL Lite. The choice among these varieties for representing a knowledge base will eventually affect the expressiveness of the ontology and whether or not reasoning algorithms will be able to guarantee completeness and/or decidability. A comprehensive study of OWL language constructs, its varying representation schemes and flavors is presented in the W3C OWL guide [6].

For the sake of explaining formal computational model of a ontology, $O$, in simple terms, we reproduce the following definition provided by Ehrig & Sure in [12]:

$$O := (C, H_C, R_C, H_R, I, R_I, A)$$

"An ontology $O$ consists of the following. The concepts $C$ of the schema are arranged in a subsumption hierarchy $H_C$. Relations $R_C$ exist between concepts. Relations (Properties) can also be arranged in a hierarchy $H_R$. Instances $I$ of a specific concept are interconnected by property instances $R_I$ . Additionally, one can define axioms $A$ which can be used to infer knowledge from already existing one." [12]

With a formal Description Logic based knowledge representation scheme and support of complex reasoners that can perform inference on the knowledge specified within an ontology, the uses of ontologies have broadened from the purely theoretical inquiry initially carried out within the area of Artificial Intelligence to encompass practical applications by domain experts across heterogeneous fields. Ontologies have been integrated into fields like biomedicine (for example, the OBO Foundry ontologies[2]), biology (for example, the Gene Ontology[3]), business modeling (for example, the Customer Complaint Ontology[4]) etc. for knowledge representation and reasoning tasks. Osterwalder & Pigneur discuss and present a generic ontology model for representing e-business issues and their interdependencies in a company's business model [31]. The Gene Ontology Consortium has been involved in building a structured, controlled vocabulary and classification of gene and gene product attributes across several domains of molecular and cellular biology [16]. Likewise, the TOVE project aims to represent the shared semantics of enterprises into an ontology [13]. All such attempts of using ontologies for representation and reasoning over domain data have proved beneficial. For example, Aroyo et al. [4] discuss how ontology based approaches could benefit user adaptive systems in terms of enhancing personalized access to multimedia content presentation while Kuo et al. [22] assert that ontology driven data mining in finding the association rules in a medical domain yields more meaningful results than naive mining techniques.

[2] `www.obofoundry.org`
[3] `http://anil.cchmc.org/Bio-Ontologies.html`
[4] `http://www.jarrar.info/CContology/`

In general, Mizoguchi et al. [30] identified some of the applications of ontologies. These include:

- As a common vocabulary for communication among distributed agents
- As a conceptual schema underlying relational data bases
- As a repository of information for a user of a certain knowledge base
- As a tool to standardize terminologies, concept meanings, target objects and tasks
- As a tool for semantic transformation of databases across heterogeneous conceptual schemas
- As a tool for reusing knowledge from a knowledge base
- As a tool for reorganizing knowledge in a knowledge base

An interesting aspect of ontology engineering in recent years has been in the field of computational linguistics. Here, as with other fields mentioned earlier, the usual approach is to use ontologies for domain modeling. In doing so, the objective is to model known linguistic phenomena (under the scope of study) as concepts and the possible associations between the phenomena as relations between those concepts. Those concepts and relations are then organized into suitable hierarchies of inheritance, yielding an ontology that captures and encodes the linguistic knowledge under study. Some examples of such domain ontologies in linguistics include PAPEL[5](which is a lexical ontology for Portuguese language), GOLD[6](which aims at building standard ontology for descriptive linguistics) and SENSUS[7](which is an ontology designed to represent linguistic taxonomy). Such domain ontologies are intended to eventually facilitate in knowledge sharing of linguistic data and in developing expert systems capable of linguistic analysis [43]. Körner & Brumm [21] discuss how an ontology based dialogue system could help a requirement analysts (in his job of ascertaining the exact needs of a software customer) in checking linguistic defects (ambiguous, faulty or inaccurate statements) in the textual specification of requirements and Knoth et al. [20] present the use of multilingual domain ontologies to support cross-language retrieval & machine translation.

A less conventional approach of utilizing ontologies in the computational linguistics field is in the area of Natural Language Generation (NLG). In this approach, an ontology is considered as a formal knowledge repository which can be utilized as a resource for NLG tasks. The objective, then, is to generate a linguistically interesting and relevant descriptive text summarizing parts or all of the concisely encoded knowledge within the given ontology. It has been

[5]http://www.linguateca.pt/papel/
[6]http://linguistics-ontology.org/
[7]http://www.isi.edu/natural-language/projects/ONTOLOGIES.html

argued that ontologies contain linguistically interesting patterns in choice of the words they use for representing knowledge and this in itself makes the task of mapping from ontologies to natural language easier [28]. It is along this line of thought that this thesis builds upon. We aim at utilizing ontologies for the sake of NLG and seek to justify the motive and rationale for doing so. Further, we will identify a set of generic questions that are suitable to be asked concerning an input ontology. The logically structured manner of knowledge organization within an ontology enables to perform reasoning tasks like Consistency checking, Concept Satisfiability, Concept Subsumption and Instance Checking. These types of logical inferencing actions will motivate us in proposing a set of Natural Language based questions which can be asked. NLG, in turn, will be applied to derive descriptive texts as answers to such queries. This will eventually help us in implementing a simple natural language based Question Answering system guided by robust NLG techniques that act upon ontologies.

In our work, ontologies written in OWL DL is taken into account because of the reasoning completeness they ensure. The reasoning capability, in turn, will help in deciding the nature/pattern of NLG activity in our research (explained in detail in chapter 4 and 5).

## 2.2 Question Answering

As noted in the previous section, the present work views the task of generating a description of some fragment of an ontology as a task of generating an answer to respond to a question. Hence, although the present work differs from standard question-answering research, it is useful to give an overview of the rationale behind Question Answering (QA), to identify the points of connection with the present work. Question Answering is the task of automatically deriving an answer to a question posed in natural language. A good definition of a QA system is given by Denicia-Carral, as follows: "A Question Answering system is an information retrieval application whose aim is to provide inexperienced users with flexible access to information, allowing them to write a query in natural language and obtaining not a set of documents that contain the answer, but the concise answer itself." [10]

Question Answering has been a long researched field in computational linguistics. Early QA techniques relied on a small structured database to retrieve facts and possibly rephrased parts or all of the syntactic structure in the query to produce answer sentences with a placeholder to fit in the retrieved fact. However, significant progress has been made in QA techniques and now we have better systems that are able to act upon a huge and heterogeneous range of corpus data, including the World Wide Web. We have observed some successful commercial implementations of QA techniques like START[8], Watson[9],

[8] http://start.csail.mit.edu/

[9] http://www-03.ibm.com/innovation/us/watson/

Wolfram|Alpha[10] etc.

Question Answering is in itself a broad area of research and involves a multitude of activities in Information Retrieval (IR) techniques, linguistic analysis, Natural Language Understanding (NLU) and NLG or even speech processing. Some or all of these activities may become relevant to a given QA system, depending upon the scope and application context of such systems. Complex QA systems that try to learn both the set of possible questions and their respective answers from the given text corpora, incorporate IR, NLU and NLG techniques on a large scale. IR would be required for retrieving facts from the corpus, NLU would be required for semantic interpretation of those facts in order to create relevant queries and NLG would be required for generation of natural language texts as answers to those queries. On the other hand, for some other QA systems which don't need much linguistic/semantic knowledge to understand a question (because the system designer implemented a known set of predetermined questions beforehand), it may suffice to restrict the NLU activities to simple tasks of keyword spotting or skip them altogether. This strategy is similar to that adopted in the present work, whose focus is on processing domain knowledge in the ontology for generating answers to a predetermined set of queries.

A specific set of questions, referred to as factoid questions in the Question Answering domain, is concentrated on querying very specific information from the available data. This set includes the usual Wh-Questions like What, How, When, Who etc. Answers to such questions tend to seek simple facts on named entities and should ideally be concise and immediately address the query. As Martin and Jurafsky put: "The task of a factoid question answering system is thus to answer questions by finding, either from the Web or some other collection of documents, short text segments that are likely to contain answers to questions, reformatting them, and presenting them to the user." [25]

Most QA systems compute upon loosely connected data sources from a wide set of documents such as the HTML documents in the World Wide Web or some other corpus. In contrast, the purpose of this thesis is to identify methods and algorithms that can yield natural language texts from structured and tightly coupled knowledge base in ontologies as answers to a selectively chosen set of predetermined factoid questions. The concentration of this research is, thus, on NLG to produce descriptive text while trying to associate some predetermined factoid questions which can better serve as a query context for the descriptive text so generated.

[10] `http://www.wolframalpha.com/`

## 2.3 Natural Language Generation

Natural Language Generation is the task of generating natural language text suitable for human consumption from machine representation of facts which can be pre-structured in some linguistically amenable fashion, or completely unstructured. The facts may be available in terms of an unstructured collection of raw data or in terms of some structured data repository (such as a relational database or a knowledge base). A class of NLG systems that build upon unstructured collections of data are known as "data-to-text" systems [35]. Examples of such systems include SUMTIME [38] (a weather report system) and BabyTalk [1] (a medical decision support system). As these systems take unstructured data source rather than a structured data repository as their input, they actually put an extra effort in structuring the raw data themselves as a precursory step to NLG.

Contrary to the "data-to-text" systems where the input is merely a collection of "flat" data, input to NLG systems can also come from more structured and organized repositories of logical expressions. Ontologies, for example, have a logically organized arrangement of facts and statements of relationships prevailing among those facts. Inherent to the relationships specified in logical formalisms (such as an ontology) are the semantics characterizing the associations among those facts. NLG developers can, in turn, have the privilege of relying on the interpretation of those associations to convey messages while generating natural language texts. Additionally, with logical expressions, formal models for computing inferences (from existing knowledge) usually exist and thus it is often quite possible to expect additional/better "content" arising from such inferencing tasks to augment the otherwise flat NLG output. Both these aspects of targeting NLG activities around systems which consume logical expressions make it an interesting avenue of research.

The motivation behind NLG is guided by several objectives. Adorni & Zock, in [2] listed the following applications areas of NLG technology:

- Story Generation/Narration
- Explanation Generation
- Reports & Paraphrases
- Intelligent help & tutoring systems
- Intelligent interfaces for Expert Systems and Databases
- Machine Translation
- Multi-party Discourse

NLG systems evolved from template based text generation systems in restricted domains. Template based systems are NLG systems in which text generation is

achieved by filling in the empty gaps within the predetermined linguistic surface structures with suitable information computed from the input data source. Well formed output results when all such gaps have been filled in [42]. A simple yet popular example of purely template based NLG systems is the mail-merge feature available with emailing applications.

A widespread and alternative approach to purely template based text generation has been in terms of organizing the generation task into several phases of linguistically motivated processing modules; each module in turn helping to shape the non-linguistic input to some intermediary representation of linguistic structure until a desired natural language text is output. Each of the modules is usually backed up by some formal models of grammar rules that carry out a specific task of linguistic computation such as lexicalisation, morphology generation, syntactic arrangement etc. It is this modular aspect of processing which makes such systems contrast with the template based systems that use a direct and predefined mapping between input and output.

In the past, template based systems have been criticized for being too restrictive to the application domain, inflexible to changes and incapable of expressing linguistically complex phenomena (like aggregation, discourse planning etc.) and thus being an inferior NLG practice compared to the modular one. However, Van Deemter et al. [42] argue that modern template systems, in fact, can implement complex recursive templates (and not just the simpler templates like that mentioned earlier for mail-merge task) which allow for fitting other template structures within them and hence are arguably as good as modular approaches in the generation of quality natural language text.

Early examples of successful NLG systems include some commercial applications like FOG (Goldberg et al. [17]) which produced textual descriptions of weather forecast and SPOTLIGHT (Anand & Kahn [3]) which produced a textual description of market analysis based on retail sales data. Over the past couple of decades, NLG systems have become steadily more robust and sophisticated, various stages in the NLG development architecture have been better understood and better results have been obtained. More recent NLG systems include applications like PERSONAGE (Mairesse & Walker [23]), which produces natural language text describing personality traits of humans based on facts obtained from psychology experiments and the ARNS system (Bethem et al. [7]) which produces automated textual narratives about water levels, coastal currents and other meteorological and oceanographic data at US ports from the real-time data obtained in graphical or tabular form etc.

Reiter & Dale [37] present a reference architecture for NLG systems. The architecture has been a common standard for contemporary NLG systems. It is reproduced here in figure 2.1.

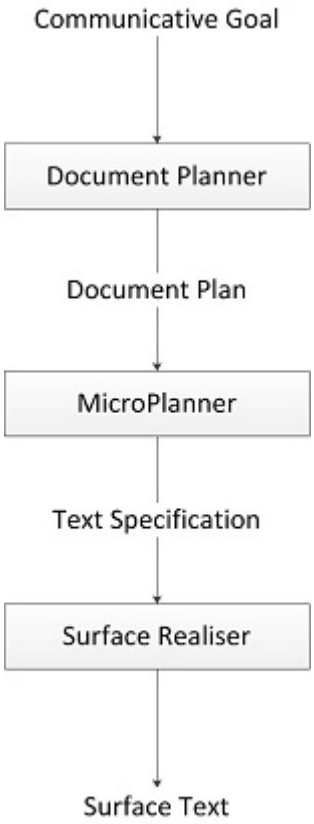


Figure 2.1: NLG system architecture as proposed by Reiter & Dale [37]

As depicted in figure 2.1, a NLG system consists of a pipeline of the following 3 processing units:

- Document Planner : The Document Planner takes the input data source and performs following two tasks:
  - Content Determination: Here, the contents from the input that are relevant and make a suitable response to the specified communicative goal of the NLG system are identified. Content Determination is a crucial task in any NLG system because it limits the scope of the NLG system in terms of what it can express. Importantly, content determination must be carried out in relation to the communicative goals which the particular NLG system has to meet. For instance, in our case, the communicative goals are, effectively, the ones expressed by the factoid questions we have identified – when we ask the question "Describe (What is) $X$?", we want the description of the concept and when we ask the question "What are the varieties of $X$?", we want to state the varieties of X etc.
  - Document Structuring: Once all such relevant contents are determined, the next step is to organize them into groups. The idea behind classifying the contents into different groups is to plan an initial order of the contents in the way that they would eventually be arranged while producing the final output of the NLG system.

  Since the nature of input and communicative goals of each NLG system is different from the other, the task of Content Determination and Document Structuring becomes specific to the NLG system under development. Hence, there is little or no general guideline that a developer can follow to accomplish these steps in his/her system. For the Document Structuring task, however, there some theories such as the Rhetorical Structure Theory [24] proposed by Mann and Thompson which discuss approaches in making relationships between different segments of the document structure explicit, even though their application differs widely from one domain to another. Usually, one has to rely on the domain specific knowledge of the input data and expert's help, as well as psycholinguistic modeling patterns to describe the domain in order to come up with a successful implementation of the Document Planner.

  The output from the Document Planner is a document plan; usually a tree representation of the content. Each node of the tree represents a group of the contents and the leaves under each node represent the contents that were classified to be under that group.

- Microplanner : The MicroPlanner takes the document plan and carries out the following tasks:
  - Lexicalisation: This involves deciding how specific elements of the non-linguistic content will help in forming a particular syntactic structure for the output text to be generated. In simpler terms, it is the task of determining what parts of the content will map to noun, verb , adjective etc. and how those words could serve as subject, verb, object, prepositional phrase or other complex syntactic structure for the output text to be generated.
  - Generating Referring Expressions (GRE): The task of generating referring expression has been a widely studied field of NLG. GRE involves in generating referring expressions (noun phrases) which can serve as a surrogate to identify an entity being described in the text. GRE is an important issue to NLG because NLG deals with producing descriptive text describing entities in a discourse. A common feature of natural languages is that they use referring expressions to uniquely identify an entity – both when the entity is first mentioned in the discourse (Initial Reference) or when the entity is subsequently referred to after it has been introduced once in the discourse (Subsequent Reference). This feature is highly desirable with any NLG system.
  - Aggregation: Aggregation is the task of identifying the contents that could be grouped together while producing a sentence or a paragraph. From the document plan, under each group, a set of contents that can communicate several pieces of information at once are identified and aggregated.

    The nature of the contents present within the document plan will guide a developer in performing the tasks of Lexicalisation and Aggregation. Apart from quite a few general rules available (such as the rule of conjunction reduction for aggregation which states that any two sentences with the same subject, say "X VP1" and "X VP2" always produce an aggregated sentence, say "X VP1 and VP2"), there are some aspects of these tasks which are highly domain dependent and the developer may therefore acquire active participation of experts or may seek to model the system based on some empirical studies of how people use natural language in the particular domain being studied. GRE, on the other hand, has been a widely studied area and there are now a variety of general algorithms that a developer can implement in order to generate referring expressions.

  The output from the MicroPlanner is a textual specification of the contents. Usually, it means that the nodes and leaves of the tree obtained in the document plan are now deep syntactic structures of sentences. Each of the processing units in the MicroPlanner – Lexicalisation, GRE and Aggregation – model the contents in the document plan in terms of their

linguistic functions and properties and thus leaves in the textual specification are now a representation of the partially complete sentence structure.

- Surface Realiser: The job of the Surface Realiser is to accept the text specification produced by the MicroPlanner as it's input and then utilize the syntactic structures present within it to generate actual text in natural language. Surface Realisation has been the most well understood aspect of NLG [36] and there are now, several software packages which can take care of the task of Surface Realisation for a developer. An example of such a package, which we shall also use during our work, is the simpleNLG system [15].

### 2.3.1 NLG with Ontologies

With reference to the issue of generating natural language text from ontology, which is the concern of this thesis, several related works have been carried out in the NLG community. Pan and Mellish [32] discuss the work behind answering questions like *What is X?* and *How is X different from Y?* based on the subsumption hierarchy information present in the ontology. Mellish & Pan further extend their work in [27] by taking into account non-standard inference in description logic, which they refer to as *Natural Language Directed Inference (NLDI)* for Content Determination. The NLDI is their attempt at organizing a sequence of logical formulas so as to best fit a computational interpretation of the Gricean Maxims of Quality, Quantity, Relevance and Manner in generating texts from ontologies; they consider linguistically motivated functions like aggregation, disaggregation, elimination of disjunctions, average number of sentences in a description etc. Bontcheva [8] presents the ONTOSUM system which models its ontology to text generation in accordance with the device profile being used by the user (such as mobile phone, the usual Web browser etc.) to view the text. Galanis & Androutsopoulos [14] present the NaturalOWL system, a multilingual ontology to text generation system, which produces natural language text from linguistically annotated ontologies. The authors argue that annotating ontologies with linguistic information helps in generating high quality multilingual text; accordingly, they design templates, one for each of the natural languages targeted by the system, which are eventually filled up with annotated information (domain-dependent linguistic information corresponding to the concept and relation names in the ontology) to generate multilingual descriptions.

Other relevant activities have been on building ontologies using NLG based authoring tools, as in [33]; utilizing ontologies to store facts about a domain (for example, a neonatal baby care unit) and later deploying reasoning over the facts in order to produce descriptive texts as in [1]; generation of multilingual text from ontologies using discourse strategies mapping them to the Grammatical Framework (GF) [34] based abstract grammar representations as in [9] etc.

## 2.4 Contributions of present work

Our work shall focus on deploying the NLG architecture in order to generate natural language text as answers to a set of predetermined factoid questions that can be asked upon the knowledge base in the ontology. We seek to innovate in this art from the existing systems in several ways. First, we proceed to obtain descriptions of concepts in the ontology rather than individuals, which is common with existing systems like ONTOSUM. Second, we identify a set of heuristics based on OWL constructs that are intended to be general enough to be of use in the Content Determination module of any ontology-to-text NLG system. Identifying such a generic set of heuristics which serve to provide "content", beforehand, is in contrast to and perhaps more modular and flexible than the approach adopted by Mellish & Sun [26], where they address the Content Determination problem by implementing a best first search strategy for axioms in the graph based representation of ontology. Further, we associate the results of reasoning in order to yield up a description of a concept in an ontology – that is we use both the explicitly stated knowledge and implicitly inferred knowledge about a concept to generate a descriptive text about it. This provides greater scope for content selection than with the NaturalOWL system, which only conveys definitions that is explicit in a concept. Additionally, we identify a generic set of factoid questions that can be posed upon ontologies and target our NLG endeavors towards generating answers to those questions. These questions implicitly serve to specify the communicative goal(s) of our NLG system. Finally, we model our Micro Planning and Realisation tasks to suit the results of empirical surveys carried out with English-speaking experimental participants who are non-experts when ontologies are concerned, on how they would like to put logical statements, similar to those present within an ontology, into natural language text.

# Chapter 3

# Retrieving Answers to Factoid Questions via Ontologies

## 3.1 Domain and Scope of work

Our task of generating text from ontologies will be motivated by the context of generating descriptive text as answer to a predetermined set of factoid questions about the ontology. As mentioned in section 2.4, we shall aim to produce descriptive text for concepts present in the ontology.

Predetermining a set of questions to be asked enables us to restrict our scope of work to Natural Language generation. As the nature of questions is previously known, there is no need for our system to work in terms of understanding the query itself and this enables us to focus on the task of appropriately modeling our NLG architecture to generate appropriate answers to each question type. We put an initial effort on identifying a set of suitable factoid questions which are relevant to be asked upon the ontology and then focus on implementing a full NLG system that can yield answers to such queries.

Our NLG approach to generating answers based on ontologies is guided by two motivations:

1. How could a general architecture be developed to identify and retrieve a suitable set of facts from any input ontology that can serve to build up textual answers (in English) to our set of factoid questions posed upon the concepts in the input ontology?

2. How could the set of facts so retrieved be linguistically processed in order to form natural language text (in English) that accurately expresses the same idea as presented in a logical fashion within the given ontology?

## 3.2 Choice of Ontology

We chose to base our work on the well-known Pizza ontology[1]. The pizza ontology is "an example ontology that contains all constructs required for the various versions of the Pizza Tutorial run by Manchester University" [11]. The decision to choose the pizza ontology for our task is based on various benefits that the Pizza ontology has to offer. First, the ontology is modeled after a commonplace and well-known class of entity, the Pizza. This will helps us in presenting our output describing Pizzas in an intuitive way to experts and non-experts alike. For such a commonplace object, our users could immediately consume the complex ideas of logical structure within an ontology, such as open world reasoning, boolean operations between concepts, restrictions and quantifications etc., once we present them in natural language text. Also, users could rely on their general knowledge about Pizza in order to provide feedback on whether or not our system had arrived at a valid conclusion[2]. Second, since the Pizza tutorial was designed to be a comprehensive summary on building ontologies for beginners, it includes most of the constructs and features of the OWL DL language. Hence, it will help us in ensuring that our NLG system has wide coverage in the terms of the types and nature of OWL DL axioms it can interpret. Third, the ontology is nicely modeled around defining concepts and subconcepts, arranging them in suitable hierarchy of multiple inheritance and enriching concepts by relating them to other concepts via OWL properties. This is an excellent opportunity for us to work with since we want to produce descriptive text describing concepts in a given ontology rather than individuals. Additionally, the ontology is written in OWL DL – a popular flavor of the OWL language which ensures reasoning completeness (meaning that all entailments are guaranteed to be computed by a reasoner) and decidability (meaning that all such entailment computations will finish within a finite amount of time) – and this will facilitate us in extracting implicit knowledge from the ontology in order to augment our output text.

The Pizza ontology serves only as a reference ontology for the sake of our work. Thus, our NLG system is not restricted just to the Pizza ontology itself. Instead, we utilize the Pizza ontology as an opportunity to come up with a generalized set of techniques that could be deployed over a broad examples of ontologies written in OWL DL.

[1]The latest version of Pizza ontology is available at `http://www.co-ode.org/ontologies/pizza/2007/02/12/pizza.owl`

[2]This would be true if the ontology was modeled to describe the domain it represents, accurately in terms of real world knowledge, which in case of the Pizza ontology is true.

## 3.3 Choice of Factoid Questions

We determined the following set of factoid questions to be asked upon the concepts present within an ontology:

1. Describe (What is) $X$?
2. How to identify a $X$?
3. What are the varieties of $X$?

where $X$ refers to any concept in the ontology except the top (**owl:Thing**) and the bottom (**owl:Nothing**) concept.

The above mentioned set of questions was determined based on the premise that ontologies are used to model domain vocabularies [5] and the aim in building our system is to produce a textual description of the concepts expressing that vocabulary. As we shall describe in Chapter 4, the question "What is X" was taken to be the most generic of the three questions type, which encompasses the other two, since the definition of a concept involves both ways – techniques of identifying it as well as determining its varieties. Hence, our NLG architecture was developed primarily with the "Describe (What is) $X$?" question in mind; answers to the other two were obtained by a relatively simple modification of the content selection and Micro Planning procedures.

## 3.4 Determination of Answer Pattern

We conducted a survey in order to find out how people would like to organize a known set of facts about some concept while describing concepts in natural language (English). The survey was based on presenting a set of facts representing a concept– resembling the set of facts describing concepts in an ontology – and asking people to come up with a description of that concept in English. We observe the pattern in which users put the facts while describing the concept and use similar pattern to structure our output. We shall discuss the nature of survey and observations made from it, in detail, in chapter 5.

# Chapter 4

# NLG in Answer Preparation

## 4.1 NLG Architecture – Overview

We follow the generic NLG architecture, presented in figure 2.1 above, for our task of generating answers to factoid questions. The pipeline structure guides us in organizing our work into sequential processing phases. Figure 4.1 below outlines the various processing steps we carry out under each phase of the NLG pipeline and we shall discuss our activities in detail throughout this chapter.

## 4.2 Retrieving data from Ontology

As with any NLG system, we begin with the task of identifying input to be fed to the NLG architecture. Our input comes from the contents of the "pizza.owl" file. In the file, facts pertaining to concepts are organized as statements (axioms) authored in RDF syntax (RDF, for sake of simplicity, can be understood as a variant of XML). We wrote a java program utilizing the open source OWL API[1] to retrieve all such axioms present within the ontology. The OWL API implements a RDF/XML parser that can read the contents of the file and output OWL Abstract Syntax based statements representing the axioms. Further, it provides a java based programming interface to reasoners (reasoners are software that can infer implicit facts from a set of explicitly stated facts available for a given ontology), such as Hermit[2], which enables us to access the implicit knowledge base from within our programming environment. The OWL Abstract Syntax based statements are more neatly formatted than that of the native RDF syntax of the ontology and thus will facilitate the processing of the axioms. For example, the RDF/XML based representation of the axiom stating that Pizza

[1]The OWL API is available at `http://owlapi.sourceforge.net/`
[2]The Hermit Reasoner is available at `http://hermit-reasoner.com/`

is a subclass of Food looks like:

```
<owl:Class rdf:about="#Pizza">
    <rdfs:label xml:lang="en">Pizza</rdfs:label>
    <rdfs:subClassOf>
                <owl:Class rdf:about="#Food"/>
    </rdfs:subClassOf>
</owl:Class>
```

while the OWL Abstract Syntax retrieved for the same axiom looks like:

```
SubClassOf(<Pizza> <Food>)
```

which is more intuitive, cleaner and easier for further processing.

We classify the available set of axioms in the ontology into the following categories and retrieve all of the axioms corresponding to each category.

1. Subsumers
   (a) Stated Subsumers with Named Concept
   (b) Stated Subsumers with Property Restriction
   (c) Implied Subsumers with Named Concept
2. Equivalents
   (a) Stated Equivalents with Property Restriction
   (b) Stated Equivalents with Enumeration
   (c) Stated Equivalents with Cardinality Restriction
   (d) Stated Equivalents with Set Operator
   (e) Implied Equivalents with Named Concept
3. Disjoints
4. Siblings

Subsumer axioms are the axioms which state that a concept (child concept) inherits properties from some other concept (parent concept) in the ontology. Typically, such axioms begin with the word “SubClassOf”. For a given concept, its subsumer can either be a named concept in the ontology or an anonymous concept (an anonymous concept is an unnamed concept which represents a new set of individuals that can be obtained after the specified property restrictions are exercised over the specified named concept in the ontology). Further, in

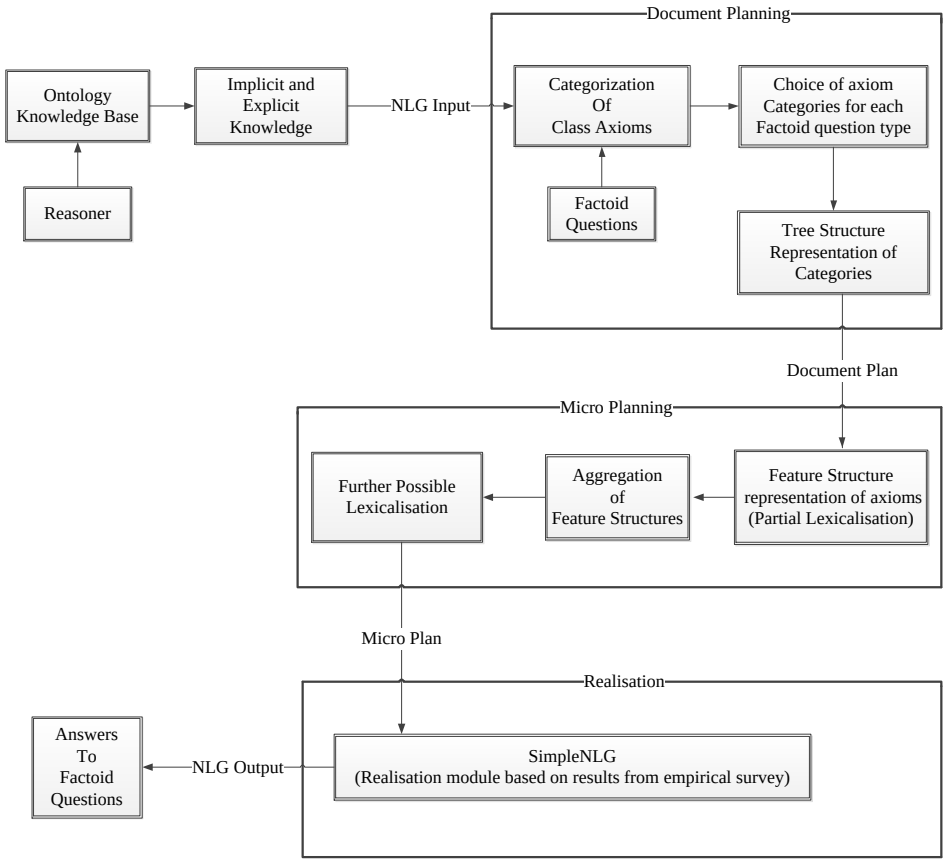


Figure 4.1: Processing steps within the generic NLG pipeline

addition to the explicitly stated subsumers (both named and anonymous type) for a given concept in the ontology, we use the reasoner to infer its additional named subsumers. An example of each category of subsumer axioms is shown below.

- Stated Subsumers with Named Concept :

$$\text{SubClassOf(<Margherita> } \underbrace{\text{<NamedPizza>}}_{\text{Named Concept}}\text{)}$$

- Stated Subsumers with Property Restriction :

$$\text{SubClassOf(<Margherita> } \underbrace{\text{ObjectSomeValuesFrom(<hasTopping> <TomatoTopping>)}}_{\text{Property Restriction}}\text{)}$$

- Implied Subsumers with Named Concept :

$$\text{SubClassOf(<Margherita> } \underbrace{\text{<CheeseyPizza>}}_{\text{Named Concept}}\text{)}$$

Equivalent axioms are the axioms which state that the concepts involved describe exactly the same set of individuals. Typically, such axioms begin with the word "EquivalentClasses". For a given concept, its equivalent concept can either be a named concept in the ontology or an anonymous concept. Here, the anonymous concepts can be defined in terms of property restrictions, cardinality restrictions, enumeration or set operations over some named concepts in the ontology. An example of each category of equivalent axioms is shown below.

- Stated Equivalents with Property Restriction :

$$\text{EquivalentClasses(<SpicyTopping> ObjectIntersectionOf(<PizzaTopping> } \underbrace{\text{ObjectSomeValuesFrom(<hasSpiciness> <Hot>)))}}_{\text{Property Restriction}}$$

- Stated Equivalents with Enumeration :

$$\text{EquivalentClasses(<Country> ObjectIntersectionOf(<DomainConcept> } \underbrace{\text{ObjectOneOf(<America> <England> <France>)))}}_{\text{Enumeration}}$$

- Stated Equivalents with Set Operator :

$$\text{EquivalentClasses(<VegetarianTopping> ObjectIntersectionOf(<PizzaTopping> } \underbrace{\text{ObjectUnionOf(<Cheese> <Nut> <Fruit>)))}}_{\text{Set Operator}}$$

- Implied Equivalents with Named Concept :

$$\text{EquivalentClasses(<SpicyPizzaEquivalent> } \underbrace{\text{<SpicyPizza>}}_{\text{Named Concept}} \text{ )}$$

Disjoint axioms are the axioms which state that the concepts involved have no individuals in common. Typically, such axioms begin with the word "DisjointClasses". An example of such an axiom is shown below:

- Disjoints:

$$\text{DisjointClasses(} \underbrace{\text{<Napoletana> <Parmense>}}_{\text{Disjoint Concepts}} \text{)}$$

There is no standard category of OWL axioms for Siblings. Hence, such construct is not directly available as an axiom in the ontology itself. We refer to concepts as being sibling of each other when they are classified as subconcepts under the same parent concept, i.e., if a concept X has children Y and Z, then Y is a sibling of Z and vice versa (also Y and Z are siblings of themselves). Further, since a concept can be classified as subconcept of various concepts in the ontology (due to multiple inheritance; either stated by the author or implied via reasoning), we identify the siblings of a given concept under its various superconcepts. Accordingly, we use the reasoner to infer the siblings of a given concept and structure a custom-defined syntax for representing the information, as shown below.

- Siblings:

$$\text{SiblingsOf(<SloppyGiuseppe>} \underbrace{\text{<MeatyPizza>}}_{\text{Context}}$$
$$\underbrace{\text{<Parmense,AmericanHot,LaReine,SloppyGiuseppe,PolloAdAstra,American>}}_{\text{Sibling Concepts}}\text{)}$$

As just described, the siblings of a concept can vary depending upon its superconcept; so we can have multiple statements describing the siblings for the same concept but under its various superconcepts. We have accordingly identified siblings of each concept under its various superconcepts, referred to as context in the above example.

## 4.3 Document Planning

Our Document Planning task comprises of following 2 processing units.

### 4.3.1 Content Determination

The task of Content Determination is an issue of open discussion for many NLG systems. Because each NLG system differs from the others in terms of its goal and implementation model, there are no general guidelines to follow, although some theories of the rhetorical structure of text have achieved widespread recognition (for example, Mann and Thompson [24]). Mellish and Sun [26] distinguish two broad categories of content determination problems – the "top-down problems" and the "bottom-up" problems. The "top-down problems" need to identify specific contents that can address a specific goal while the "bottom-up problems" have a more diffuse goal of identifying contents that can produce a general expository or descriptive text. In our case, the set of factoid questions that we have determined are themselves the specifications of the communicative goal (as we argued earlier when discussing Content Determination in section 2.3) of our system; thus conforming to the "top-down" approach and since we aim to generate descriptive text about concepts in the ontology, our system also adheres to the "bottom-up" model of content determination problems. Addressing the problem, the categories of axioms we retrieved in section 4.2 will serve as "content" for our NLG system. In particular, we identify the categories of axioms that will provide the "content" for generation of sentences to serve as answers for each of the factoid questions we discussed in section 3.3.

- For the factoid question, "Describe (What is) $X$?", all the categories of axioms we described in section 4.2 will serve as "content" for the generation of answers. As such, the answer to this type of factoid question, for a given concept X, will provide users with the most detailed description of various facts pertaining to that concept in the given ontology.
- For the factoid question, "How to identify a $X$?", the category of equivalent axioms, both stated and implied, when present will serve as "content" for the generation of answers. The equivalent axioms associated with a concept express logical conditions (also referred to as sufficient conditions in the literature), which when fulfilled, are sufficient to distinguish the concept from the others. Our system interprets the semantics behind those logical conditions into natural language text so as to come up with an answer to this type of factoid question.
- For the factoid question, "What are the varieties of $X$?", we view the varieties of a given concept X as essentially its subconcepts (both stated and implied). Accordingly, we use the subconcepts information available for the concept to generate answer to this factoid question.

### 4.3.2 Document Structuring

Here, we work on organizing the axioms so as to facilitate a coherent and efficient future processing. As is common with contemporary NLG techniques, we adopt the tree data structure for our representation scheme. The set of axioms retrieved is organized into hierarchical fashion as depicted in figure 4.2. The tree represents our initial plan of organizing the axiom categories into suitable representation scheme for processing. With subsequent processing phases on the NLG architecture pipeline, the nodes of the tree will shrink and grow and their contents will get modified; we shall discuss the corresponding operations which will effect such changes.

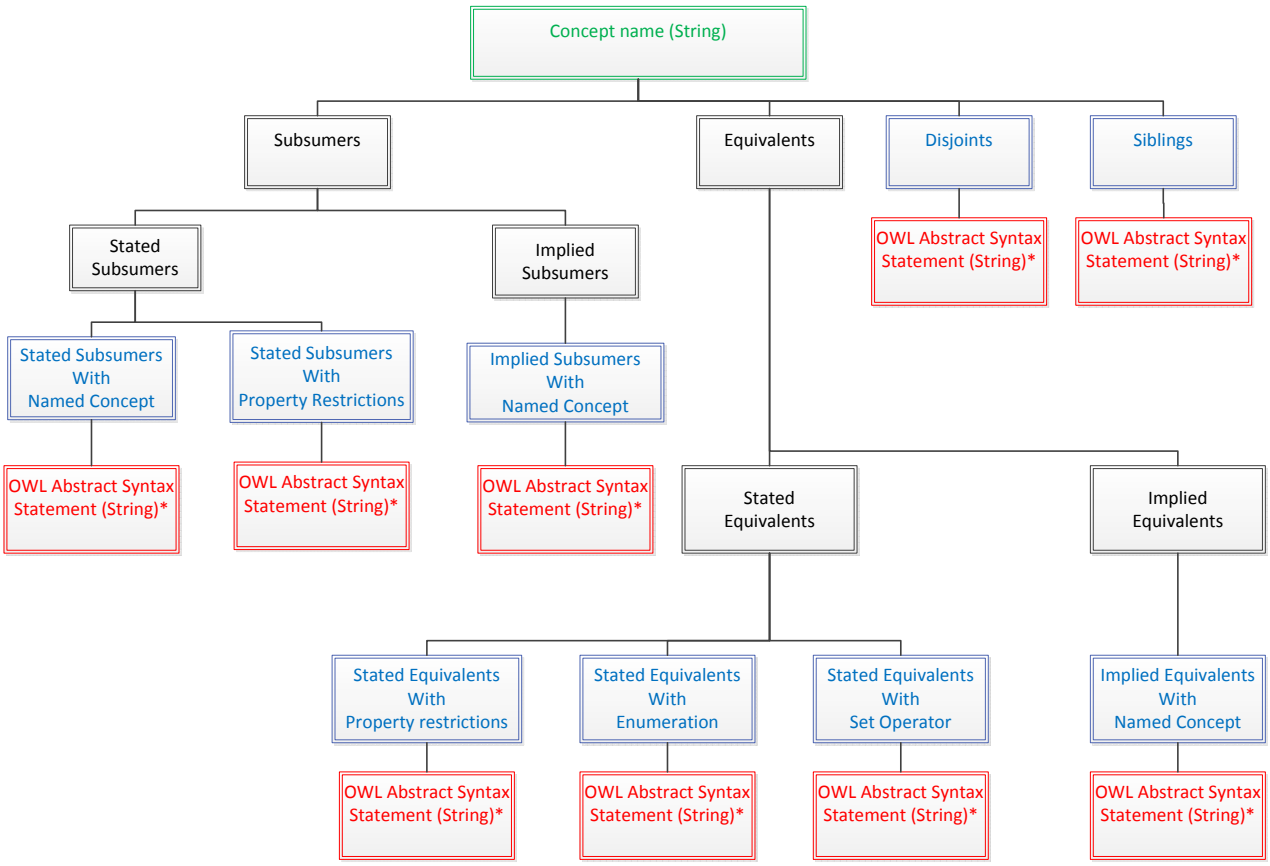


Figure 4.2: Document Structure

As shown in figure 4.2, the root node of the tree (presented in green box) represents the concept for which the NLG architecture is being modeled to yield answers to the set of factoid questions mentioned in section 3.3. Each of the subterminal nodes (presented in blue box) of the tree stands for one of the categories of axioms mentioned in section 4.2 and under such node, are zero or more leaves (presented in red box); each one representing an axiom (in OWL Abstract Syntax, as discussed in section 4.2) falling under that category for that concept in the ontology. It is possible that a concept doesn't have any leaves under certain subterminal nodes – for example, a concept may not have any Stated Equivalent axioms in the ontology. All other intermediate nodes (presented in black box) in the tree serve to represent the organization of the terminal nodes into hierarchical divisions of the tree branching; conforming to our classification of the category of axioms.

At this phase of NLG architecture, we represent both the concept name at the root and the axioms at the leaves simply as strings.

## 4.4 Micro Planning

Our Micro Planning activity comprises the following 3 processing units.

### 4.4.1 PreLexicalisation

In NLG, Lexicalisation is the task of identifying lexical items (words of natural language) that will serve to build up natural language sentences. The lexical items are vocabulary for the sentence. During the Lexicalisation phase, we work on identifying lexical items that will help us in mapping the factual knowledge within each category of OWL axioms to natural language text.

We divide our Lexicalisation task into 2 sub stages: Pre-Lexicalisation phase followed by Lexicalisation Proper. In the Pre-Lexicalisation stage, we come up with an initial plan of sentence structure and preliminary choice of lexical items for each category of the statements presented in section 4.2. As we shall discuss below, it is better to postpone part or whole of the lexicalisation task for some category of the statements until after a subsequent processing module in the Micro Planning activity – the Aggregation phase. We use the term Lexicalisation Proper to refer to this postponed activity of lexicalisation and shall discuss it subsequently in section 4.4.3. In this section, we discuss the activities carried out during the Pre-Lexicalisation stage.

It is easy to notice that the statements retrieved in section 4.2 are semi linguistic in nature. In each example, we have statements which contain concepts and relations that are either legitimate words of English (eg: Pizza, Food, Country etc.) or are concatenations of legitimate words of English (eg: hasBase, MeatyPizza, PizzaTopping etc.). Based on similar observations, Sun & Mellish

[41] argued that it is feasible to derive lexical items from the semi linguistic concept and relation names in the ontology itself. We follow this idea and implement strategies, which are discussed below, in generating lexical items for our task. Further, the semantics of the predicates being used in the statements guide us on identifying what linguistic roles such lexical items play in the output sentence to be generated. Let us consider the following statement, for example:

```
SubClassOf(<Margherita> ObjectSomeValuesFrom(<hasTopping>
                                             <TomatoTopping>))
```

The concept name "Margherita" is a legitimate word of English and the concept name "TomatoTopping" is formed by concatenation of two English words "Tomato" & "Topping". Likewise, the relation "hasTopping" is also formed by concatenation of two legitimate words – "has" & "Topping". Additionally, the semantics of the predicate "SubClassOf" guides us in mapping the concept "Margherita" to the subject and the concept "TomatoTopping" to the object of the sentence that we intend to verbalize from the axiom. Similarly, the relation "hasTopping" is a good candidate for the verb in the sentence to be generated; since relationships in OWL act as binders between concepts, they can serve to identify the linguistic roles of subject and object in the output sentence to be generated. We exploit such semantic information to plan the output sentence structure for each category of the statements presented in section 4.2.

We plan the output sentence structures as feature structure based representations. We design the feature structure to be a matrix based representation whereby we identify a set of features (usually motivated by linguistic roles such as Subject, Verb and Object) and associate the extracted lexical items as values for those features in the feature structure. We generate distinct feature structures for each of the categories of statements presented in section 4.2, respectively. Each feature structure depicts our initial plan for framing output sentences under that category of axioms and is internally maintained as a HashMap data structure.

With regards to the task of identifying lexical items to represent concepts, the concept names that are legitimate words of English are directly approved as lexical items for our task; we shall refer to such concept names as "**Simplified Concept Name**". For other concept names which are formed by concatenation of two or more English words, we work to derive their possible breakdown into Simplified Concept Names, which will then serve as lexical items.

The strategy that we apply in deriving Simplified Concept Name from complex concept name and which is widely used during our PreLexicalisation phase is to identify the superconcept of the given concept whose name is present as the terminally concatenated string in the given concept name. Since there can be multiple superconcepts for a concept in the ontology (either stated or inferred via reasoning), it is possible to check for such a possibility among a number of superconcepts. The superconcept that satisfies such requirements is then chosen

and designated as "**Best Parent**". This strategy is based on our observation of the pattern in which the classes and subclasses are usually organized in a hierarchy within the ontology. A sample hierarchical arrangement of classes within the ontology is depicted in figure 4.3.

In figure 4.3, we can see that the concept "CheeseyPizza" has multiple superconcepts (either stated or implied) – "Pizza", "Food" & "DomainConcept". The concept name "CheeseyPizza" is made up of a concatenation of the words "Cheesey" and "Pizza", of which the terminally concatenating word "Pizza" is also the name of one of its superconcepts – the "Pizza" concept. As per our discussion above, we then designate the concept "Pizza" as the **"Best Parent"** for the concept "CheeseyPizza".

Once we identify the "BestParent" for a given concept, we generate its **"Simplified Concept Name"** by creating a new string which is essentially a substring of its name whereby the name of the "Best Parent" concept that was terminally present is removed. For the example of "CheeseyPizza" concept discussed above, the resulting value for **"Simplified Concept Name"** is "Cheesey"; obtained by removing its terminally concatenated word "Pizza" – the name of its **"Best Parent"** concept.

Since any ontology hierarchy is usually designed to arrange concepts in such a fashion that subconcepts specialize their superconcept, it is logical to assume that the **"Simplified Concept Name"** retrieved after processing the concept against its **"Best Parent"** name serves as a lexical item modifying (specializing) that **"Best Parent"** name. This allows us to model the features describing a concept (in our feature structure representation) in terms of base form and its modifier. The base form is set to the value of the **"Best Parent"** name and the modifier is set to the value of the **"Simplified Concept Name"**. For example, for the concept "CheeseyPizza" , we have its base form set to "Pizza" and its modifier set to "Cheesey". We will refer this strategy of generating the base form and modifier for a given concept as **"Concept Lexicalisation Algorithm"** in the rest of this chapter.

In case the **"Best Parent"** can't be computed for a given concept, its base form is set to the concept name itself and the modifier feature is removed. For example, the superconcepts (stated or implied) available for the concept "CheeseTopping" in the pizza ontology are "PizzaTopping", "VegetarianTopping", "Food" & "DomainConcept"; none of which make its **"Best Parent"**. Thus, for the concept "CheeseTopping", we have its base form set to "CheeseTopping" and there is no modifier.

Below, we present the feature structures that we design for each category of axioms. The feature structure for each category displays a list of features with their corresponding values.


Figure 4.3: An Example Ontology Concept Hierarchy

**Feature Structure for Stated Subsumers with Named Concept**

For this category of axioms, we designate feature structures consisting of features "Subject", "Object" & "ObjectModifier". From the statement expressing the axiom, we identify the concepts that will serve the linguistic roles of Subject and Object in the output sentence to be generated. Once such Subject and Object concepts are determined, we carry out the **"Concept Lexicalisation Algorithm"** over the Object concept to generate values for the **"Object"** and **"ObjectModifier"** features in the feature structure. The value for the **"Subject"** feature is simply set to the name of the concept representing the Subject. We present an example feature structure below.

| **Stated Subsumers with Named Concept** | |
|---|---|
| Statement Prototype | SubClassOf(<Concept_X> <Concept_Y>) |
| Feature Structure Prototype | [**Subject** : *Concept_X*<br>**Object** : *BestParent of Concept_Y*<br>**ObjectModifier** : *Simplified Concept_Y*] |
| Example Statement | SubClassOf(<Cajun> <NamedPizza>) |
| Example Feature Structure | [**Subject** : Cajun<br>**Object** : Pizza<br>**ObjectModifier** : Named] |

Table 4.1: F.S. for Stated Subsumers with Named Concept, Case:I

However, there can be Object concepts (like "Pizza" in the example presented in table 4.2 below), whose **"Best Parent"** can't be computed (the superconcepts for the concept "Pizza" are "Food" & "DomainConcept"; none of which make it's **"Best Parent"**). In such cases, we only have the "Object" feature in the feature structure whose value is simply set to the name of the concept.

| **Stated Subsumers with Named Concept** | |
|---|---|
| Statement Prototype | SubClassOf(<Concept_X> <Concept_Y>) |
| Feature Structure Prototype | [**Subject** : *Concept_X*<br>**Object** : *Concept_Y*] |
| Example Statement | SubClassOf(<VegetarianPizza> <Pizza>) |
| Example Feature Structure | [**Subject** : VegetarianPizza<br>**Object** : Pizza] |

Table 4.2: F.S. for Stated Subsumers with Named Concept, Case:II

**Feature Structure for Stated Subsumers with Property Restriction**

For this category of axioms, we simply seek to correspond the concepts and relation present within the axiom statement to a set of features – **Subject**, **Verb** and **Object**. Also, this category of axioms can be further specialized into 3 sub categories. They are:

- Universal Restrictions

  A universal restriction axiom restricts a concept (Subject) participating in a relation (Verb) over a concept (Object) or over a set of concepts (Objects). When a set of concepts are involved, we represent them in a comma separated values (CSV) fashion. An example for each case is presented below.

| **Stated Subsumers with Property Restriction(*Universal*)** | |
|---|---|
| Statement Prototype | SubClassOf(<Concept_X> ObjectAllValuesFrom (<Relation_A> <Concept_Y>)) |
| Feature Structure Prototype | [**Subject** : *Concept_X*<br>**Verb** : *Relation_A*<br>**Object** : *Concept_Y*] |
| Example Statement | SubClassOf(<RealItalianPizza> ObjectAllValuesFrom(<hasBase> <ThinAndCrispyBase>)) |
| Example Feature Structure | [**Subject** : RealItalianPizza<br>**Verb** : hasBase<br>**Object** : ThinAndCrispyBase] |

Table 4.3: F.S. for Stated Subsumers with Universal Restriction, Case:I

| **Stated Subsumers with Property Restriction(*Universal*)** | |
|---|---|
| Statement Prototype | SubClassOf(<Concept_X> ObjectAllValuesFrom (<Relation_A> ObjectUnionOf (<Concept_Y>$^+$ ))) |
| Feature Structure Prototype | $\begin{bmatrix} \textbf{Subject} : \textit{Concept\_X} \\ \textbf{Verb} : \textit{Relation\_A} \\ \textbf{Object} : \textit{CSV of concepts in } (\textit{Concept\_Y})^+ \end{bmatrix}$ |
| Example Statement | SubClassOf(<Rosa> ObjectAllValuesFrom (<hasTopping> ObjectUnionOf (<GorgonzolaTopping> <MozzarellaTopping> <TomatoTopping>))) |
| Example Feature Structure | $\begin{bmatrix} \textbf{Subject} : \text{Rosa} \\ \textbf{Verb} : \text{hasTopping} \\ \textbf{Object} : \text{GorgonzolaTopping}, \\ \qquad\qquad \text{MozzarellaTopping}, \\ \qquad\qquad \text{TomatoTopping} \end{bmatrix}$ |

Table 4.4: F.S. for Stated Subsumers with Universal Restriction, Case:II

- Existential Restrictions

  We design the following feature structure to represent the category of existential restriction axioms which restrict a concept (Subject) participating in a relation (Verb) over another concept (Object).

| **Stated Subsumers with Property Restriction(*Existential*)** | |
|---|---|
| Statement Prototype | SubClassOf (<Concept_X> ObjectSomeValuesFrom (<Relation_A> <Concept_Y>)) |
| Feature Structure Prototype | $\begin{bmatrix} \textbf{Subject} : \textit{Concept\_X} \\ \textbf{Verb} : \textit{Relation\_A} \\ \textbf{Object} : \textit{Concept\_Y} \end{bmatrix}$ |
| Example Statement | SubClassOf (<Margherita> ObjectSomeValuesFrom (<hasTopping> <TomatoTopping>)) |
| Example Feature Structure | $\begin{bmatrix} \textbf{Subject} : \text{Margherita} \\ \textbf{Verb} : \text{hasTopping} \\ \textbf{Object} : \text{TomatoTopping} \end{bmatrix}$ |

Table 4.5: F.S. for Stated Subsumers with Existential Restriction

- HasValue Restrictions

  Similarly, we have the following feature structure to represent the category of HasValue restrictions which restrict a concept (Subject) participating in a relation (Verb) over a particular instance (Object) of a concept.

| **Stated Subsumers with Property Restriction(*HasValue*)** | |
|---|---|
| Statement Prototype | SubClassOf (<Concept_X> ObjectHasValue (<Relation_A> <Concept_Y>)) |
| Feature Structure Prototype | [ **Subject** : *Concept_X*<br>**Verb** : *Relation_A*<br>**Object** : *Concept_Y* ] |
| Example Statement | SubClassOf (<Napoletana> ObjectHasValue (<hasCountryOfOrigin> <Italy>)) |
| Example Feature Structure | [ **Subject** : Napoletana<br>**Verb** : hasCountryOfOrigin<br>**Object** : Italy ] |

Table 4.6: F.S. for Stated Subsumers with HasValue Restriction

**Feature Structure for Implied Subsumers with Named Concept**

Implied subsumers for a concept are obtained by reasoning. An implied subsumer inferred for a concept can in turn be equivalent to some other concepts in the ontology. Based on this observation, we present a feature structure representation for two cases – the first, where the implied subsumer is simply a named concept in the ontology and the second, where the subsumer concept is a named concept which in turn is equivalent to one or more other concepts in the ontology.

In both cases, we seek to compute lexical items for the features **"Object"** and **"ObjectModifier"** based on our **"Concept Lexicalisation Algorithm"**. In case where the subsumer is simply a named concept in the ontology, the task of computing lexical items for those features is similar to the one we described earlier for the category of "Stated Subsumers with Named Concept" axioms. An example for cases, where the **"Best Parent"** can (for example, the concept "ThinAndCrispyPizza" in table 4.7) and can't (for example, the concept "SpicyTopping" in table 4.8) be computed is shown separately below.

| **Implied Subsumers with Named Concept** | |
|---|---|
| Statement Prototype | SubClassOf (<Concept_X> <Concept_Y>) |
| Feature Structure Prototype | [ **Subject** : *Concept_X*<br>**Object** : *BestParent of Concept_Y*<br>**ObjectModifier** : *Simplified Concept_Y* ] |
| Example Statement | SubClassOf (<RealItalianPizza> <ThinAndCrispyPizza>) |
| Example Feature Structure | [ **Subject** : RealItalianPizza<br>**Object** : Pizza<br>**ObjectModifier** : ThinAndCrispy ] |

Table 4.7: F.S. for Implied Subsumers with Named Concept, Case:I

| **Implied Subsumers with Named Concept** | |
|---|---|
| Statement Prototype | SubClassOf (<Concept_X> <Concept_Y>) |
| Feature Structure Prototype | [ **Subject** : *Concept_X*<br>**Object** : *Concept_Y* ] |
| Example Statement | SubClassOf (<HotGreenPepperTopping> <SpicyTopping>) |
| Example Feature Structure | [ **Subject** : HotGreenPepperTopping<br>**Object** : SpicyTopping ] |

Table 4.8: F.S. for Implied Subsumers with Named Concept, Case:II

However, in cases where the subsumer concept is also equivalent to some other named concept in the ontology (for example, the subsumer concept "SpicyPizza" is also equivalent to another named concept "SpicyPizzaEquivalent" in table 4.9 below), the task become a bit more involved. First, since the subsumer contains equivalent concepts, our **Best Parent** will be the non empty string retrieved from either one of those equivalent concepts. Next, we generate the possible **"Simplified Concept Name"** for each of those equivalent concepts by processing them against the **"Best Parent"**, in turn. Then we set the **ObjectModifier** feature's value by arranging those **"Simplified Concept Name"**s in an order where the ones for which the simplification was not successful are placed earlier followed by the ones for which the simplification was successful (such order promotes readability in the output sentence to be generated). For example, we present the values for the **"Object"** and **"ObjectModifier"** features in table 4.9 below, based on the fact that the **"Best Parent"** for the concept "SpicyPizzaEquivalent" could not be computed but the **"Best Parent"** computed for the concept "SpicyPizza" was "Pizza".

| **Implied Subsumers with Named Concept** | |
|---|---|
| Statement Prototype | SubClassOf (<Concept_X> <Concept_Y=Concept_Z=...>) |
| Feature Structure Prototype | **Subject** : *Concept_X*<br>**Object** : *BestParent*<br>**ObjectModifier** : *Simplified Concept_Y* |
| Example Statement | SubClassOf (<Cajun> <SpicyPizza=SpicyPizzaEquivalent>) |
| Example Feature Structure | **Subject** : Cajun<br>**Object** : Pizza<br>**ObjectModifier** : SpicyPizzaEquivalent /Spicy |

Table 4.9: F.S. for Implied Subsumers with Named Concept, Case:III

**Feature Structure for Stated Equivalents with Property Restriction**

Equivalent class axioms are the ones which define a given concept (defined concept) in terms of some other concept (defining concept) in the ontology. Thus it is logical to assume that the defined concept is a special variant of the defining concept. Along this line of thought, we identify the features **"Subject"** and **"SubjectDescriptor"** to be present in the feature structure corresponding to this category of axioms. The value for the **"Subject"** is set to the defined concept's name and the value for the **"SubjectDescriptor"** feature is set to the defining concept's name. Eventually, the value set for **"SubjectDescriptor"** feature will be mapped to a sentence of the form "X is a SubjectDescriptor" in the output. Other features in this feature structure include **Verb** and **Object**.

Also, this category of axioms can be specialized into further three sub categories. They are:

- Universal Restrictions

  A universal restriction axiom restricts a concept (Subject) participating in a relation (Verb) over a single concept (Object) or over a set of concepts (Objects). When a set of concepts are involved, we represent them in a comma separated values (CSV) fashion. An example for each case is presented below.

| **Stated Equivalents with Property Restriction(*Universal*)** | |
|---|---|
| Statement Prototype | EquivalentClasses (<Concept_X> ObjectIntersectionOf (<Concept_Y> ObjectAllValuesFrom (<Relation_A> <Concept_Z>))) |
| Feature Structure Prototype | [ **Subject** : *Concept_X*<br>**SubjectDescriptor** : *Concept_Y*<br>**Verb** : *Relation_A*<br>**Object** : *Concept_Z* ] |
| Example Statement | EquivalentClasses (<ThinAndCrispyPizza> ObjectIntersectionOf (<Pizza> ObjectAllValuesFrom (<hasBase> <ThinAndCrispyBase>))) |
| Example Feature Structure | [ **Subject** : ThinAndCrispyPizza<br>**SubjectDescriptor** : Pizza<br>**Verb** : hasBase<br>**Object** : ThinAndCrispyBase ] |

Table 4.10: F.S. for Stated Equivalents with Universal Restriction, Case:I

| **Stated Equivalents with Property Restriction(*Universal*)** | |
|---|---|
| Statement Prototype | EquivalentClasses (<Concept_X> ObjectIntersectionOf (<Concept_Y> ObjectAllValuesFrom (<Relation_A> ObjectUnionOf (<Concept_Z>$^+$))) ) |
| Feature Structure Prototype | [ **Subject** : *Concept_X*<br>**SubjectDescriptor** : *Concept_Y*<br>**Verb** : *Relation_A*<br>**Object** : *CSV of concepts in* $(Concept\_Z)^+$ ] |
| Example Statement | EquivalentClasses (<VegetarianPizzaEquivalent2> ObjectIntersectionOf (<Pizza> ObjectAllValuesFrom (<hasTopping> ObjectUnionOf (<CheeseTopping> <FruitTopping> <HerbSpiceTopping> <NutTopping> <SauceTopping> <VegetableTopping>)))) |
| Example Feature Structure | [ **Subject** : VegetarianPizzaEquivalent2<br>**SubjectDescriptor** : Pizza<br>**Verb** : hasTopping<br>**Object** : CheeseTopping,<br>FruitTopping,<br>HerbSpiceTopping,<br>NutTopping,<br>SauceTopping,<br>VegetableTopping ] |

Table 4.11: F.S. for Stated Equivalents with Universal Restriction, Case:II

- Existential Restrictions

  An existential restriction axiom restricts a concept (Subject) participating in a relation (Verb) simply over a concept (Object) or over a concept (Object) which is further restricted with regards to its participation with some other concepts via some other relation in the ontology. In the later case, where the Object is further restricted, we designate a new feature structure as the value for the **Object** feature in the main feature structure. An example for each case is shown below.

| **Stated Equivalents with Property Restriction(*Existential*)** | |
|---|---|
| Statement Prototype | EquivalentClasses (<Concept_X> ObjectIntersectionOf (<Concept_Y> ObjectSomeValuesFrom (<Relation_A> <Concept_Z>))) |
| Feature Structure Prototype | [ **Subject** : *Concept_X*<br>**SubjectDescriptor** : *Concept_Y*<br>**Verb** : *Relation_A*<br>**Object** : *Concept_Z* ] |
| Example Statement | EquivalentClasses (<MeatyPizza> ObjectIntersectionOf (<Pizza> ObjectSomeValuesFrom (<hasTopping> <MeatTopping>))) |
| Example Feature Structure | [ **Subject** : MeatyPizza<br>**SubjectDescriptor** : Pizza<br>**Verb** : hasTopping<br>**Object** : MeatTopping ] |

Table 4.12: F.S. for Stated Equivalents with Existential Restriction, Case:I

| **Stated Equivalents with Property Restriction(*Existential*)** | |
|---|---|
| Statement Prototype | EquivalentClasses (<Concept_X> ObjectIntersectionOf (<Concept_Y> ObjectSomeValuesFrom (<Relation_A> ObjectIntersectionOf (<Concept_Z> ObjectSomeValuesFrom (<Relation_B> <Concept_V>))))) |
| Feature Structure Prototype | $\begin{bmatrix} \textbf{Subject} : \textit{Concept\_X} \\ \textbf{SubjectDescriptor} : \textit{Concept\_Y} \\ \textbf{Verb} : \textit{Relation\_A} \\ \textbf{Object} : \begin{bmatrix} \textbf{Subj\_1} : \textit{Concept\_Z} \\ \textbf{Verb\_1} : \textit{Relation\_B} \\ \textbf{Obj\_1} : \textit{Concept\_V} \end{bmatrix} \end{bmatrix}$ |
| Example Statement | EquivalentClasses (<SpicyPizzaEquivalent> ObjectIntersectionOf (<Pizza> ObjectSomeValuesFrom (<hasTopping> ObjectIntersectionOf (<PizzaTopping> ObjectSomeValuesFrom (<hasSpiciness> <Hot>))))) |
| Example Feature Structure | $\begin{bmatrix} \textbf{Subject} : \text{SpicyPizzaEquivalent} \\ \textbf{SubjectDescriptor} : \text{Pizza} \\ \textbf{Verb} : \text{hasTopping} \\ \textbf{Object} : \begin{bmatrix} \textbf{Subj\_1} : \text{PizzaTopping} \\ \textbf{Verb\_1} : \text{hasSpiciness} \\ \textbf{Obj\_1} : \text{Hot} \end{bmatrix} \end{bmatrix}$ |

Table 4.13: F.S. for Stated Equivalents with Existential Restriction, Case:II

- HasValue Restrictions

  Similarly, we have the following feature structure to represent the category of HasValue restrictions which restrict a concept (Subject) participating in a relation (Verb) over a particular instance (Object) of a concept.

| **Stated Equivalents with Property Restriction(*HasValue*)** | |
|---|---|
| Statement Prototype | EquivalentClasses (<Concept_X> ObjectIntersectionOf (<Concept_Y> ObjectHasValue (<Relation_A> <Concept_Z>))) |
| Feature Structure Prototype | [ **Subject** : *Concept_X*<br>**SubjectDescriptor** : *Concept_Y*<br>**Verb** : *Relation_A*<br>**Object** : *Concept_Z* ] |
| Example Statement | EquivalentClasses (<RealItalianPizza> ObjectIntersectionOf (<Pizza> ObjectHasValue (<hasCountryOfOrigin> <Italy>))) |
| Example Feature Structure | [ **Subject** : RealItalianPizza<br>**SubjectDescriptor** : Pizza<br>**Verb** : hasCountryOfOrigin<br>**Object** : Italy ] |

Table 4.14: F.S. for Stated Equivalents with HasValue Restriction

**Feature Structure for Stated Equivalents with Enumeration**

An enumeration axiom defines a given concept (Subject) by exhaustively enumerating all of its instances (Object). The feature structure we designed to represent enumeration axioms is shown below.

| **Stated Equivalents with Enumeration** | |
|---|---|
| Statement Prototype | EquivalentClasses (<Concept_X> ObjectIntersectionOf (<Concept_Y> ObjectOneOf (<Concept_Z>$^{+}$))) |
| Feature Structure Prototype | [ **Subject** : *Concept_X*<br>**SubjectDescriptor** : *Concept_Y*<br>**Object** : *CSV of concepts in* $(Concept\_Z)^{+}$ ] |
| Example Statement | EquivalentClasses (<Country> ObjectIntersectionOf (<DomainConcept> ObjectOneOf (<America> <England> <France> <Germany> <Italy>))) |
| Example Feature Structure | [ **Subject** : Country<br>**SubjectDescriptor** : DomainConcept<br>**Object** : America,<br>England,<br>France,<br>Germany,<br>Italy ] |

Table 4.15: F.S. for Stated Equivalents with Enumeration

**Feature Structure for Stated Equivalents with Cardinality Restrictions**

Cardinality restriction axioms state the fact that a given concept (**Subject**) can participate for a specified (min, max or exact) number of times in a given relation (**Verb**) with another concept (**Object**). Thus, in addition to the usual features, we have two additional features – **CardinalityType** and **CardinalityValue** in the feature structures representing this category of axioms. Below, we present a generic prototype with an example feature structure specific to the "min" type cardinality.

| **Feature Structure for Stated Equivalents with Cardinality** | |
|---|---|
| Statement Prototype | EquivalentClasses (<Concept_X> ObjectIntersectionOf (<Concept_Y> ObjectMinCardinality (Integer_Value <Relation_A> <Concept_Z>))) |
| Feature Structure Prototype | **Subject** : *Concept_X*<br>**SubjectDescriptor** : *Concept_Y*<br>**Verb** : *Relation_A*<br>**Object** : *Concept_Z*<br>**CardinalityType** : *Min/Max/Exact*<br>**CardinalityValue** : *Integer_Value* |
| Example Statement | EquivalentClasses (<InterestingPizza> ObjectIntersectionOf (<Pizza> ObjectMinCardinality (3 <hasTopping> owl:Thing))) |
| Example Feature Structure | **Subject** : InterestingPizza<br>**SubjectDescriptor** : Pizza<br>**Verb** : hasTopping<br>**Object** : owl: Thing<br>**CardinalityType** : Min<br>**CardinalityValue** : 3 |

Table 4.16: F.S. for Stated Equivalents with Cardinality Restrictions

**Feature Structure for Stated Equivalents with Set Operator**

Two types of Set Operator were encountered in the ontology – **ObjectUnionOf** and **ObjectComplementOf**. We present the feature structure for each of those respectively, below.

| **Stated Equivalents with Set Operator(ObjectUnionOf)** | |
|---|---|
| Statement Prototype | EquivalentClasses (<Concept_X> ObjectIntersectionOf (<Concept_Y> ObjectUnionOf (<Concept_Z>$^{+}$))) |
| Feature Structure Prototype | **Subject** : *Concept_X*<br>**SubjectDescriptor** : *Concept_Y*<br>**Object** : *CSV of concepts in* $(Concept\_Z)^{+}$ |
| Example Statement | EquivalentClasses (<VegetarianTopping> ObjectIntersectionOf (<PizzaTopping> ObjectUnionOf (<CheeseTopping> <FruitTopping> <HerbSpiceTopping> <NutTopping> <SauceTopping> <VegetableTopping>))) |
| Example Feature Structure | **Subject** : VegetarianTopping<br>**SubjectDescriptor** : PizzaTopping<br>**Object** : CheeseTopping,<br>FruitTopping,<br>HerbSpiceTopping,<br>NutTopping,<br>SauceTopping,<br>VegetableTopping |

Table 4.17: F.S. for Stated Equivalents with ObjectUnionOf Operator

| **Stated Equivalents with Set Operator(ObjectComplementOf)** | |
|---|---|
| Statement Prototype | EquivalentClasses (<Concept_X> ObjectIntersectionOf (<Concept_Y> ObjectComplementOf (<Concept_Z>))) |
| Feature Structure Prototype | **Subject** : *Concept_X*<br>**SubjectDescriptor** : *Concept_Y*<br>**Object** : *Concept_Z* |
| Example Statement | EquivalentClasses (<NonVegetarianPizza> ObjectIntersectionOf (<Pizza> ObjectComplementOf (<VegetarianPizza>))) |
| Example Feature Structure | **Subject** : NonVegetarianPizza<br>**SubjectDescriptor** : Pizza<br>**Object** : VegetarianPizza |

Table 4.18: F.S. for Stated Equivalents with ObjectComplementOf Operator

**Feature Structure for Implied Equivalents with Named Concept**

The implied equivalent named concepts for a given concept are obtained by reasoning. We map the concept names present within the axiom to a set of features – **Subject** (holding the name of the given concept) and **Object** (holding the name of equivalent concept for the given concept). An example representing the feature structure for this category of statements is presented below.

| **Implied Equivalents with Named Concept** | |
|---|---|
| Statement Prototype | EquivalentClasses (<Concept_X> <Concept_Y>) |
| Feature Structure Prototype | [**Subject** : *Concept_X*<br>**Object** : *Concept_Y*] |
| Example Statement | EquivalentClasses (<SpicyPizzaEquivalent> <SpicyPizza>) |
| Example Feature Structure | [**Subject** : SpicyPizzaEquivalent<br>**Object** : SpicyPizza] |

Table 4.19: F.S. for Implied Equivalents with Named Concept

**Feature Structure for Disjoints**

Here too, we map the concept names present within the axiom to a set of features – **Subject** and **Object**. An example representing the feature structure for this category of statements is presented below.

| **Disjoints** | |
|---|---|
| Statement Prototype | DisjointClasses (<Concept_X> <Concept_Y>) |
| Feature Structure Prototype | [**Subject** : *Concept_X*<br>**Object** : *Concept_Y*] |
| Example Statement | DisjointClasses (<Napoletana> <Parmense>) |
| Example Feature Structure | [**Subject** : Napoletana<br>**Object** : Parmense] |

Table 4.20: F.S. for Disjoints

**Feature Structure for Siblings**

As discussed earlier in section 4.2, we identify the siblings of a concept with reference to its superconcept (referred to as *Context* in our earlier example). We compute the **"Best Parent"** for the superconcept and carry out our **"Concept Lexicalisation Algorithm"** to generate lexical items for the **"Context"** and **"ContextModifier"** features. Further, we assign the **Object** feature with a list (as CSV) of sibling concept names identified for the given concept in the **Subject** feature. An example feature structure is shown below.

| **Siblings** | |
|---|---|
| Statement Prototype | SiblingsOf (<Concept_X> <Concept_Y> <(Concept_Z)$^+$>) |
| Feature Structure Prototype | **Subject** : *Concept_X*<br>**Context** : *Best Parent of Concept_Y*<br>**ContextModifier** : *Simplified Concept_Y*<br>**Object** : *CSV of concepts in* $(Concept\_Z)^+$ |
| Example Statement | SiblingsOf (<SloppyGiuseppe> <MeatyPizza> <Parmense,AmericanHot,LaReine,SloppyGiuseppe>) |
| Example Feature Structure | **Subject** : SloppyGiuseppe<br>**Context** : Pizza<br>**ContextModifier** : Meaty<br>**Object** : Parmense,<br>AmericanHot,<br>LaReine,<br>SloppyGiuseppe |

Table 4.21: F.S. for Siblings

### 4.4.2 Aggregation

Aggregation is the task of grouping two or more simple structures to generate a single sentence, a frequent phenomenon in natural languages. For example, instead of having sentences like "It is a Friday Afternoon" & "It is a lovely Afternoon", it is preferable to have a single sentence, like "It is a lovely Friday Afternoon", generated by aggregating the facts present in both of the sentences.

A lot of work on aggregation has focused on aggregating syntactic structures. By contrast, in our work, we identify aggregation of possible feature structures from the PreLexicalisation phase based on the semantics of what they express. The main utility of doing aggregation at this pre-syntactic level is that much of our aggregation decisions depend on the semantics – we don't want to risk aggregating structures that are syntactically eligible but belong to different con-

cepts, for example. We use a notation of array (shown as [ ] brackets) to describe and process the aggregated contents (so as to contrast it to the CSV list that we have used earlier for representing a collection of items) in discussion of our aggregation strategies of feature structures for each category of axioms, below.

**Aggregation of Stated and Implied Subsumers with Named Concept**

A concept may have one or more named concepts that are explicitly stated to be its subsumers in the ontology hierarchy. Also, via reasoning, it is possible that a given concept has more named subsumers. Since both of these categories of axioms describe the set of named subsumers (either stated or implied), we seek to aggregate them into a single feature structure. In order to aggregate all such named subsumers available for a given concept, first, we retrieve all the feature structures corresponding to the stated as well as implied category of subsumers with named concept axioms in its document plan tree. Next, we generate a new array and populate it with the value assigned to the **ObjectModifier** feature in each of those feature structures retrieved. Subsequently, we generate a new feature structure consisting of features **Subject**, **Object** and **ObjectModifier**. This new feature structure will have its value for the **Subject** feature set to the common value of the **Subject** feature of the retrieved feature structures; value for the **Object** feature set to the common value of the **Object** feature of the retrieved feature structures and value for the **ObjectModifier** feature set to the newly formed array. The feature structures are now said to be aggregated and for further processing, we only consider this new feature structure yielded after aggregation. However, for the concepts for which no more than one named subsumer can be retrieved (because no more named subsumers could be inferred via reasoning), aggregation is not feasible and we preserve the single feature structure (for the stated named subsumer) obtained from the PreLexicalisation phase for further processing.

Let us consider an example. For the concept "Napoletana" in the pizza ontology, the concept "NamedPizza" is stated to be it's subsumer and the concepts "InterestingPizza", "CheeseyPizza", "RealItalianPizza" and "NonVegetarianPizza" are inferred to be its subsumers. Subsequently, during our PreLexicalisation phase, we had the following feature structures to represent them, respectively.

$$\begin{bmatrix} \textbf{Subject} : \text{Napoletana} \\ \textbf{Object} : \text{Pizza} \\ \textbf{ObjectModifier} : \text{Named} \end{bmatrix} \;\&\; \begin{bmatrix} \textbf{Subject} : \text{Napoletana} \\ \textbf{Object} : \text{Pizza} \\ \textbf{ObjectModifier} : \text{Interesting} \end{bmatrix} \;\&$$

$$\begin{bmatrix} \textbf{Subject} : \text{Napoletana} \\ \textbf{Object} : \text{Pizza} \\ \textbf{ObjectModifier} : \text{Cheesey} \end{bmatrix} \;\&$$

$$\begin{bmatrix} \textbf{Subject} : \text{Napoletana} \\ \textbf{Object} : \text{Pizza} \\ \textbf{ObjectModifier} : \text{RealItalian} \end{bmatrix} \;\&\; \begin{bmatrix} \textbf{Subject} : \text{Napoletana} \\ \textbf{Object} : \text{Pizza} \\ \textbf{ObjectModifier} : \text{NonVegetarian} \end{bmatrix}$$

Now, during the aggregation phase, the following new feature structure is generated and preserved for further processing; eliminating the above five feature structures.

$$\begin{bmatrix} \textbf{Subject} : \text{Napoletana} \\ \textbf{Object} : \text{Pizza} \\ \textbf{ObjectModifier} : [\,\text{Named},\ \text{Interesting},\ \text{Cheesey},\ \text{RealItalian},\ \text{NonVegetarian}\,] \end{bmatrix}$$

**Aggregation of Stated Subsumers with Property Restriction**

For this category of class axioms, earlier, we identified three different subcategories – the Universal Restrictions, the Existential Restrictions & the HasValue restrictions. Further, for each of these subcategories, we identified feature structures consisting of features – **Subject**, **Verb** & **Object**. Corresponding to each subcategory, one or more axioms may be present for a given concept name in an ontology. Our criteria in deciding the candidate axioms for aggregation task, under each of those subcategories, is to identify the axioms which bear the same value for the **Verb** feature in their feature structure.

Under each subcategory, we first classify the available feature structures (from the PreLexicalisation stage) into separate groups; each group representing a set of feature structures which have their **Verb** feature's value in common. There can be one or more such groups depending upon the variety of relations (**Verb**) upon which the restrictions have been specified for the given concept. Next, pertaining to each group, we designate a new array and a new feature structure. The array is populated with the values retrieved for the **Object** feature of the feature structures belonging to that group. The new feature structure is composed of the feature **Subject** set to the common **Subject** of the feature structures in the group; the feature **Verb** set to the common **Verb** of feature structures in the group and the feature **Object** set to the newly formed array in the group. The axioms are now said to be aggregated and for further processing, we only consider these new feature structures yielded after aggregation. However, for the feature structures for which aggregation is not feasible because there are no multiple restriction axioms (of the same subcategory type) specified over the same relation for the given concept, we preserve them, without changes, for further processing. Based on this strategy, we discuss below the aggregation carried out for each subcategory respectively.

- Universal Restrictions

  None of the concepts in the pizza ontology have multiple universal restriction axioms specified over the same relation. Thus, no aggregation is feasible for this subcategory of axioms, and we retain the feature structures obtained during the PreLexicalisation phase for further processing.

For sake of generality, however, we present an imaginary example below to represent aggregation of Universal restrictions – if we have the following two feature structures representing the universal restrictions for the concept "Tree" participating in the relation "hasVertex" over the concepts "AcylicVertex" (meaning that all vertices of a tree are acyclic) & "TreeVertex" (meaning that all vertices of a tree are of type tree-vertex) respectively

$$\begin{bmatrix} \textbf{Subject} : \text{Tree} \\ \textbf{Verb} : \text{hasVertex} \\ \textbf{Object} : \text{AcylicVertex} \end{bmatrix} \quad \& \quad \begin{bmatrix} \textbf{Subject} : \text{Tree} \\ \textbf{Verb} : \text{hasVertex} \\ \textbf{Object} : \text{TreeVertex} \end{bmatrix}$$

the following new aggregated feature structure would then be generated (meaning that all vertices of a tree are both acyclic and of type tree-vertex at the same time) and preserved for further processing; eliminating the above two feature structures.

$$\begin{bmatrix} \textbf{Subject} : \text{Tree} \\ \textbf{Verb} : \text{hasVertex} \\ \textbf{Object} : [\,\text{AcylicVertex},\ \ \text{TreeVertex}\,] \end{bmatrix}$$

- Existential Restrictions

An example of aggregation carried out for feature structures representing Existential Restrictions is presented below. The concept "Margherita", in the pizza ontology, has following axioms representing existential restrictions:

SubClassOf (<Margherita> ObjectSomeValuesFrom (<hasTopping> <TomatoTopping>))

&

SubClassOf (<Margherita> ObjectSomeValuesFrom (<hasTopping> <MozzarellaTopping>))

which were respectively represented via feature structures (during the PreLexicalisation phase) as:

$$\begin{bmatrix} \textbf{Subject} : \text{Margherita} \\ \textbf{Verb} : \text{hasTopping} \\ \textbf{Object} : \text{TomatoTopping} \end{bmatrix} \quad \& \quad \begin{bmatrix} \textbf{Subject} : \text{Margherita} \\ \textbf{Verb} : \text{hasTopping} \\ \textbf{Object} : \text{MozzarellaTopping} \end{bmatrix}$$

Now, during the aggregation phase, the following new feature structure is generated and preserved for further processing; eliminating the above two feature structures.

$$\begin{bmatrix} \textbf{Subject} : \text{Margherita} \\ \textbf{Verb} : \text{hasTopping} \\ \textbf{Object} : [\,\text{TomatoTopping},\ \ \text{MozzarellaTopping}\,] \end{bmatrix}$$

- HasValue Restrictions

  None of the concepts in the pizza ontology have multiple HasValue restriction axioms specified over the same relation. Thus, no aggregation is feasible for this subcategory of axioms, and we retain the feature structures obtained during the PreLexicalisation phase for further processing. When present, the multiple axioms specifying the HasValue restriction over the same relation will have an aggregated feature structure resembling the prototypes we just presented for the aggregation of universal and existential restrictions.

**Aggregation of Stated Equivalents with Property Restrictions**

Since this category of axioms *define* concepts in terms of the relations they participate in with other concepts in the ontology, usually, such definition of a concept over a given relation is stated as a single axiom expressing all the constraints rather than as a set of multiple axioms, each one expressing a separate constraint over the same relation. On a similar note, we also observe that none of the concepts, under our classification of Stated Equivalents (which, in turn, includes the subcategories Property Restrictions, Enumeration, Cardinality and Set Operators), in the pizza ontology, bear multiple sets of equivalent axioms defined over the same relation. Thus, we skip aggregation for this category of axioms and retain the feature structures generated during the PreLexicalisation phase for further processing.

**Aggregation of Implied Equivalents with Named Concepts**

For this category of axioms, from the PreLexicalisation phase, we have feature structures consisting of the feature **Subject** & **Object**. For aggregating the set of feature structures, we first generate a new array and populate it with the value of the **Object** feature from each of the feature structures retrieved. Then we generate a new feature structure consisting of the feature **Subject** set to the common value of **Subject** feature and the feature **Object** set to the newly formed array. The feature structures are now said to be aggregated and for further processing, we only consider this new feature structure yielded after aggregation. However, for the concepts for which no more than one implied Equivalent concept can be inferred, aggregation is not feasible and we preserve the feature structure obtained during the PreLexicalisation phase for further processing. For example, if we have the following two sets of feature structures representing two different implied equivalent concepts, Y & Z respectively, for the concept X:

$$\begin{bmatrix} \textbf{Subject} : \text{Concept\_X} \\ \textbf{Object} : \text{Concept\_Y} \end{bmatrix} \;\&\; \begin{bmatrix} \textbf{Subject} : \text{Concept\_X} \\ \textbf{Object} : \text{Concept\_Z} \end{bmatrix}$$

our aggregation will yield,

$$\begin{bmatrix} \textbf{Subject} : \text{Concept\_X} \\ \textbf{Object} : [\,\text{Concept\_Y},\ \text{Concept\_Z}\,] \end{bmatrix}$$

There are no concepts in the pizza ontology for which more than one equivalent concept can be inferred. Thus, no aggregation is feasible and we retain the feature structures obtained during the PreLexicalisation phase for further processing.

**Aggregation of Disjoints**

There can be zero or more disjoint concepts for a given concept; accordingly, there can be zero or more axioms stating the disjoint concepts. We aim to aggregate multiple feature structures representing those axioms. First, we build up a CSV list of concepts from the values assigned to the **Object** feature in the feature structures. Next, we form a new feature structure consisting of feature **Subject** set to the common value of the **Subject** feature of the feature structures retrieved and the feature **Object** set to the newly formed CSV list of concepts. The feature structures are now said to be aggregated and for further processing, we only consider this new feature structure yielded after aggregation. The concepts for which no more than one disjoint concept is available, aggregation is not feasible and we preserve the feature structure obtained from the PreLexicalisation phase for further processing.

Let us consider an example. For the concept "PrawnsTopping" in the pizza ontology, the concepts "MixedSeafoodTopping" and "AnchoviesTopping" are stated to be its disjoint concepts. Subsequently, for the following feature structure that we have from the PreLexicalisation phase

$$\begin{bmatrix} \textbf{Subject} : \text{PrawnsTopping} \\ \textbf{Object} : \text{MixedSeafoodTopping} \end{bmatrix} \ \& \ \begin{bmatrix} \textbf{Subject} : \text{PrawnsTopping} \\ \textbf{Object} : \text{AnchoviesTopping} \end{bmatrix}$$

we generate the following new aggregated feature structure eliminating the above two feature structures; which is then preserved for further processing.

$$\begin{bmatrix} \textbf{Subject} : \text{PrawnsTopping} \\ \textbf{Object} : [\,\text{MixedSeafoodTopping},\ \text{AnchoviesTopping}\,] \end{bmatrix}$$

**Aggregation of Siblings**

During the PreLexicalisation phase, we designed the Siblings feature structure to represent the siblings of a given concept under each of its different superconcept (**Context**). Thus, even if we have multiple feature structure representing the siblings for a given concept, all those feature structures specify the siblings of the given concept under it's different superconcepts **Context**. Hence no aggregation of this category of axioms is feasible and we preserve the feature structures from the PreLexicalisation phase for further processing.

### 4.4.3 Lexicalisation Proper

The completion of Aggregation phase opens up an opportunity to generate further lexical items; which was otherwise infeasible/unsuitable to obtain during the PreLexicalisation phase. In particular, we attempt to generate lexical items for the **Verb** features (whenever present) in the feature structures and as we shall further discuss, identifying the **Verb** lexical item will facilitate in augmenting the information pertaining to the **Object** feature in those feature structures. The need to postpone such activities until the aggregation phase has been completed stems from the fact that our Aggregation task is highly dependent on the values assigned to **Verb** features in the feature structures; they served as a criterion for judging whether an aggregation task should be carried out on the available set of feature structures or not. Thus it was desirable that those values come directly from the name (string) representing the relation in the axiom and remain "intact" throughout the aggregation phase. Let us consider an example. Suppose that a concept "X" has the following two axioms, both under the category of "Stated Subsumers with Property Restriction (Existential)".

SubClassOf (<Concept_X> ObjectSomeValuesFrom (<hasBase> <Concept_Y>))
&
SubClassOf (<Concept_X> ObjectSomeValuesFrom (<hasTopping> <Concept_Z>))

As per our strategy to generate the feature structures corresponding to these statements at the PreLexicalisation phase, we would have the following feature structures:

$$\begin{bmatrix} \textbf{Subject} : \text{Concept\_X} \\ \textbf{Verb} : \text{hasBase} \\ \textbf{Object} : \text{Concept\_Y} \end{bmatrix} \quad \& \quad \begin{bmatrix} \textbf{Subject} : \text{Concept\_X} \\ \textbf{Verb} : \text{hasTopping} \\ \textbf{Object} : \text{Concept\_Z} \end{bmatrix}$$

Now, during the aggregation phase, these two feature structures can not be aggregated as they don't meet the criteria we defined earlier for aggregation – they don't bear the same value for the feature **Verb**.

Had we instead attempted to generate lexical items for the **Verb** features during the PreLexicalisation phase itself, we would have possibly come up with the same lexical item "has" (we will soon discuss our strategy to generate lexical items from a relation name) for both of the relation names – "hasBase" and "hasTopping" and the feature structures would look like:

$$\begin{bmatrix} \textbf{Subject} : \text{Concept\_X} \\ \textbf{Verb} : \text{has} \\ \textbf{Object} : \text{Concept\_Y} \end{bmatrix} \quad \& \quad \begin{bmatrix} \textbf{Subject} : \text{Concept\_X} \\ \textbf{Verb} : \text{has} \\ \textbf{Object} : \text{Concept\_Z} \end{bmatrix}$$

In such case, the criteria for aggregation would be fulfilled and accordingly these two feature structures would have been aggregated to yield:

$$\begin{bmatrix} \textbf{Subject} : \text{Concept\_X} \\ \textbf{Verb} : \text{has} \\ \textbf{Object} : [\,\text{Concept\_Y},\ \ \text{Concept\_Z}\,] \end{bmatrix}$$

which is misleading because it groups concepts that are restricted via different relations into same group.

However, after the termination of the Aggregation phase, we already have separate groups of aggregated feature structures for separate relation names. Now, it is safe to break down those relation names into lexical items suitable for our task of natural language generation. The approach we implement in generating lexical items to represent relation names is based on our observation that relation names in an ontology are either legitimate words (Verb) of English, such as "knows", "teaches" etc. or are made up of two or more concatenating words (Verb followed by Noun or Adjective), such as "hasBase", "hasTopping" etc. We use a POS (Part of Speech) tagger, the Stanford POS Tagger[3], to incrementally look for substring which can be identified as the verb lexical item. For example, in the string representing the relation name, "hasBase", we incrementally look for substrings (such as 'h', 'ha', 'has') until the POS tagger identifies that the string "has" is a Verb and then we assign the **Verb** feature in the feature structure to value of "has".

Once the Verb is extracted from such concatenated strings, we use the remainder of the string to serve as our lexical item in augmenting the information pertaining to the **Object** feature in the feature structure. We identify a new feature named **ObjectDescriptor** to represent such lexical item in our feature structure. Additionally, we check for the possibility of reforming the existing value of the **Object** feature by removing its terminally concatenating word if the word matches the lexical item just assigned for the **ObjectDescriptor** feature in the same feature structure. For example, if we have a feature structure as follows:

$$\begin{bmatrix} \textbf{Subject} : \text{Margherita} \\ \textbf{Verb} : \text{hasTopping} \\ \textbf{Object} : \text{TomatoTopping} \end{bmatrix}$$

we first determine the Verb lexical item to be "has". Then, we set the value for the **ObjectDescriptor** feature to "topping". Finally, based on the observation that the terminally concatenating word of the value assigned to the **Object** feature is the same as the lexical item just assigned for the **ObjectDescriptor** feature, we have the opportunity to reform the value of the **Object** feature and set it to "tomato". The above feature structure would then become:

$$\begin{bmatrix} \textbf{Subject} : \text{Margherita} \\ \textbf{Verb} : \text{has} \\ \textbf{ObjectDescriptor} : \text{topping} \\ \textbf{Object} : \text{tomato} \end{bmatrix}$$

[3]Java implementation of Stanford POS Tagger is available at `http://nlp.stanford.edu/software/tagger.shtml`

Below, we discuss the changes brought about during the Lexicalisation Proper phase in the feature structures corresponding to each category of axioms.

### Lexicalisation Proper of Stated and Implied Subsumers with Named Concept

For this category of axioms, we simply preserve the feature structure obtained from the Aggregation phase for further processing.

### Lexicalisation Proper of Stated Subsumers with Property Restriction

Here we generate lexical item for the **Verb** and the **ObjectDescriptor** feature and check out for possibilities to reform the value assigned to the **Object** feature. For example, the aggregated feature structure

$$\begin{bmatrix} \textbf{Subject} : \text{Margherita} \\ \textbf{Verb} : \text{hasTopping} \\ \textbf{Object} : [\,\text{TomatoTopping},\ \ \text{MozzarellaTopping}\,] \end{bmatrix}$$

obtained after aggregation of existential restrictions on the concept "Margherita" over the relation "hasTopping" during the Aggregation phase, now becomes:

$$\begin{bmatrix} \textbf{Subject} : \text{Margherita} \\ \textbf{Verb} : \text{has} \\ \textbf{ObjectDescriptor} : \text{topping} \\ \textbf{Object} : [\,\text{Tomato},\ \ \text{Mozzarella}\,] \end{bmatrix}$$

### Lexicalisation Proper of Stated Equivalents with Property Restriction

Here too, we generate lexical items for the **Verb** and the **ObjectDescriptor** feature and check out for possibilities to reform the value assigned to the **Object** feature. For example, for the feature structure representing universal restrictions in table 4.11, we now have the following new feature structure.

$$\begin{bmatrix} \textbf{Subject} : \text{VegetarianPizzaEquivalent2} \\ \textbf{SubjectDescriptor} : \text{Pizza} \\ \textbf{Verb} : \text{has} \\ \textbf{ObjectDescriptor} : \text{topping} \\ \textbf{Object} : \text{Cheese}, \\ \qquad\qquad \text{Fruit}, \\ \qquad\qquad \text{HerbSpice}, \\ \qquad\qquad \text{Nut}, \\ \qquad\qquad \text{Sauce}, \\ \qquad\qquad \text{Vegetable} \end{bmatrix}$$

For the Existential Restriction subcategory, earlier, we prototyped two varieties of feature structure, – the first in which the restriction is simply a concept and the second in which the restriction is on a concept that is further restricted with regards to its participation with some other concepts via some other relation in the ontology. In the first case, we generate a lexical item for the **Verb** and the **ObjectDescriptor** feature and check out for possibilities to reform the value assigned to the **Object** feature. For example, for the feature structure presented in table 4.12, we now have the following new feature structure.

$$\begin{bmatrix} \textbf{Subject} : \text{MeatyPizza} \\ \textbf{SubjectDescriptor} : \text{Pizza} \\ \textbf{Verb} : \text{has} \\ \textbf{ObjectDescriptor} : \text{topping} \\ \textbf{Object} : \text{Meat} \end{bmatrix}$$

In the second case, we generate lexical items for the **Verb** and the **ObjectDescriptor** feature in both the main feature structure representing the axiom as well as in the nested feature structure representing the **Object** feature. Based on the value of the **ObjectDescriptor** feature of the main feature structure, we check for possibilities to reform the value assigned to the **Subject** feature of the nested feature structure and based on the value of the **ObjectDescriptor** feature of the nested feature structure, we check for the possibility to reform the value assigned to the **Object** feature of the nested feature structure. For example, we now have the following new feature structure for the one shown in table 4.13.

$$\begin{bmatrix} \textbf{Subject} : \text{SpicyPizzaEquivalent} \\ \textbf{SubjectDescriptor} : \text{Pizza} \\ \textbf{Verb} : \text{has} \\ \textbf{ObjectDescriptor} : \text{topping} \\ \textbf{Object} : \begin{bmatrix} \textbf{Subject} : \text{Pizza} \\ \textbf{Verb} : \text{has} \\ \textbf{ObjectDescriptor} : \text{Spiciness} \\ \textbf{Object} : \text{Hot} \end{bmatrix} \end{bmatrix}$$

For the HasValue Restriction subcategory, we obtain lexical items for the **Verb** and the **ObjectDescriptor** feature. For example, for the feature structure representing HasValue restriction in table 4.14, we have the following new feature structure.

$$\begin{bmatrix} \textbf{Subject} : \text{RealItalianPizza} \\ \textbf{SubjectDescriptor} : \text{Pizza} \\ \textbf{Verb} : \text{has} \\ \textbf{ObjectDescriptor} : \text{CountryofOrigin} \\ \textbf{Object} : \text{Italy} \end{bmatrix}$$

**Lexicalisation Proper of Stated Equivalents with Enumeration**

For this category of axioms, we simply preserve the feature structure obtained from the Aggregation phase for further processing.

**Lexicalisation Proper of Stated Equivalents with Cardinality Restriction**

We generate lexical items for the **Verb** and the **ObjectDescriptor** feature. For example, for the feature structure presented in table 4.16, we now have the following new feature structure.

$$\begin{bmatrix} \textbf{Subject} : \text{InterestingPizza} \\ \textbf{SubjectDescriptor} : \text{Pizza} \\ \textbf{Verb} : \text{has} \\ \textbf{ObjectDescriptor} : \text{topping} \\ \textbf{Object} : \text{owl: Thing} \\ \textbf{CardinalityType} : \text{Min} \\ \textbf{CardinalityValue} : 3 \end{bmatrix}$$

**Lexicalisation Proper of Stated Equivalents with Set Operator, Implied Equivalents with Named Concept, Disjoints and Siblings**

For all of these category of axioms, we simply preserve the feature structure obtained from their Aggregation phase for further processing.

## 4.5 Realisation

Realisation is the task of generating actual natural language sentences from the intermediary (syntactic) representations obtained during the Micro Planning phase. Developing an actual realisation module is outside the scope of this thesis. For a NLG developer, a number of general purpose realisation modules are available to achieve such functionality. In particular, such modules facilitate in transforming syntactic information into natural language text by taking care of various syntactic (for example, the arrangement of Subject, Verb and Object in a sentence), morphological (for example, the generation of inflected forms of words, when required, such as the plural of *child* being *children* and not *childs*) and orthographical (for example, placement of appropriate punctuation marks in the sentence, such as placing a comma to describe an aggregation of things) transformations that the contents from the Micro Planning phase need to adhere to for generating grammatically valid sentences. Realisation has been a widely studied area of NLG and there are now several wide coverage soft-

ware packages available to a developer. Some examples include FUF/SURGE[4], OpenCCG[5], SimpleNLG[6] etc. We use SimpleNLG [15] for our task because of its ease of understanding and simplicity of use. SimpleNLG also provides a java based API which can be used to access its functionality from our programming environment.

SimpleNLG has java classes which allow a programmer to specify the content (such as Subject, Verb, Object, Modifiers, Tense, Preposition phrase etc) of a sentence by setting values to the attributes in the classes. Once such attributes are set, methods can be executed to generate output sentences; the package takes care of the linguistic transformations and ensures grammatically well formed sentences. For example, to generate the sentence "The quick brown fox jumps over the lazy dog", we identify the lexical items that will serve the role of Subject, Subject modifier, Verb, Object, Object modifier etc. in the sentence and call appropriate methods that will generate the given sentence. The code listing for the example looks like:

```
/* Specify lexical items. */

NPPhraseSpec subject = nlgFactory.createNounPhrase("fox");
subject.addPreModifier("quick");
subject.addPreModifier("brown");
subject. setSpecifier ("the");

VPPhraseSpec verb = nlgFactory.createVerbPhrase("jump");
NPPhraseSpec object = nlgFactory.createNounPhrase("dog");
object. setSpecifier ("the");
object.addPreModifier("lazy");

PPPhraseSpec prepphrase = nlgFactory.createPrepositionPhrase();
prepphrase.addComplement(object);
prepphrase.setPreposition("over");

/* Call appropriate methods */

SPhraseSpec sentenceFrame = nlgFactory.createClause();
sentenceFrame.setSubject(subject);
sentenceFrame.setVerb(verb);
sentenceFrame.setComplement(prepphrase);
```

Our task during Realisation is thus to find a suitable strategy for mapping the contents from our feature structures (under each category of axioms) to appropriate attributes in SimpleNLG classes and execute proper methods to generate

[4] http://www.cs.bgu.ac.il/surge/index.html
[5] http://openccg.sourceforge.net/
[6] http://code.google.com/p/simplenlg/

the corresponding sentences. This is guided by the results of an empirical survey we carried out on identifying the way people prefer to structure facts, similar to those present for each category of axioms in section 4.2, while generating natural language sentences to describe them. We shall discuss our survey methodology and conclusions derived from the survey in section 5.1 and 5.3. Based on results of the survey, we then map our feature structures from the Micro Planning phase into suitable SimpleNLG attributes and call appropriate methods to generate sentences that will serve as response to the various types of factoid questions we identified in section 3.3. We shall present sample questions and answers in section 5.5.

# Chapter 5

# Experiments and Results

We carried out an empirical study to identify the best strategies for rendering knowledge present in ontologies to natural language text. The study was in the form of a survey comprising several test cases. Each test case was designed to convey the semantics behind a variety of class axioms (that we identified in section 4.2), by providing an example scenario (in a fashion similar to the way facts are organized in an ontology) around some commonplace concepts like pizza, student, football team etc. For each test case, we asked the participants to generate a description of the concept based entirely on the facts provided for that concept in the particular test case. The participants were free to make decisions about what they want from the available facts to express, determine what lexical items (words) they prefer to include in their statement (lexicalisation), if one or more facts could be grouped while producing such statement (aggregation) and also the syntactic, morphological and orthographical transformation (realisation) they wanted to use.

## 5.1 Survey Material

We designed a total of 8 test cases, each test case consisting of a set of facts. We put efforts in presenting each test case as a set of linguistically expressed facts (simple assertive sentences) while also flavoring it with the nature and organization pattern of facts in ontology formalisms (presenting complex concept names such as MeatyPizza, arranging the information so as to resemble the hierarchal arrangement of facts in ontologies etc.). This makes it feasible to target our survey among naive users who, solely based on the facts in test cases, can guide us in realizing similar patterns of information in ontologies. Below, we reproduce the various test cases that we presented to our participants during the survey.

Test Case I : Simplification of Concept Name (in Subject Role)

The aim of this test case is to find out if people prefer to generate base form and modifier for the concept name that serves the linguistic role of Subject in the output sentence to be generated, whenever there exists a possibility of identifying its **"Best Parent"**. For example, we provided the following facts to the participants of the survey:

Fact Set 5.1: Simplification of Concept Name (in Subject Role)

Facts:

1. The following classes of PepperTopping are known:
   a. GreenPepperTopping
   b. JalapenoPepperTopping
   c. SweetPepperTopping

and posed the question "Describe **GreenPepperTopping**".

Test Case II : Simplification of Concept Name (in Object Role), Expression of Named Subsumers and Possibilities of Aggregation

The aim of this test case is to two-folds. First, we aim to find out if people prefer to generate base form and modifier for the concept name that serves the linguistic role of Object in the output sentence to be generated, whenever there exists a possibility of identifying its **"Best Parent"**. Second, we aim to identify the popular pattern of expressing subsumer information and their possible aggregation in the output sentence. Accordingly, we provided following facts to the participants of the survey:

Fact Set 5.2: Simplification of Concept Name (in Object Role)

Facts:

1. The following classes of Pizza are known:
   a. DeliciousPizza
   b. CheeseyPizza
   c. VegetarianPizza
   d. RealItalianPizza
2. Napoletana falls under the class of CheeseyPizza.
3. Napoletana falls under the class of VegetarianPizza.
4. Napoletana falls under the class of DeliciousPizza.
5. Napoletana falls under the class of RealItalianPizza.

and posed the question "Describe **Napoletana**".

Test Case III : Expression of Named Equivalents and Aggregation

The aim of this test case is to find out how people express the idea that a given concept is logically equivalent to the other. In terms of OWL language, equivalent concepts refer to the fact that the concepts involved describe exactly the same set of instances. Accordingly, we designed our Fact Set 5.3 to present participants with the concepts defining exactly the same set of instances, as shown below.

**Fact Set 5.3: Expression of Named Equivalents and Aggregation**

Facts:

1. The following are all the known instances of WeekDays:
   a. Monday
   b. Tuesday
   c. Wednesday
   d. Thursday
   e. Friday
2. The following are all the known instances of WorkingDays:
   a. Monday
   b. Tuesday
   c. Wednesday
   d. Thursday
   e. Friday
3. The following are all the known instances of BusinessDays:
   a. Monday
   b. Tuesday
   c. Wednesday
   d. Thursday
   e. Friday

We then asked the question "Describe WeekDays. (In particular, how do you say that all of these terms - WeekDays, WorkingDays and BusinessDays express equivalent concepts?)".

Test Case IV : Expression of Universal Restriction

Universal Restrictions in OWL represent the "all or none" condition; meaning that the instances of the concept for which the universal restriction is specified upon a property X, can either **only** have relationships (via the property X) to instances of a **specific** concept or **not** have any relationship (via the property X) to any instances of any concepts at all.

We modeled our Fact Set 5.4 to represent the universal restriction over the property "has free access to" specified for the instances of the concept "EUCitizen" in relation to instances of the concept "EUCountries", as shown below.

**Fact Set 5.4: Expression of Universal Restriction**

Facts:

1. The following classes of Citizen are known:
   a. EUCitizen
   b. AmericanCitizen
2. The following classes of Countries are known:
   a. EUCountries
   b. NorthAmericanCountries
3. The following are all the known instances of EUCountries:
   a. Italy
   b. Norway
   b. Spain
   b. Hungary
4. The following are all the known instances of NorthAmericanCountries:
   a. United States
   b. Canada
   b. Mexico
5. Instances of EUCitizen can have free access to instances of EUCountries only.
6. However, some instances of EUCitizen may not have any free access rights (because the particular instance of EUCitizen was a criminal, for example).

We then asked the question "Describe EUCitizen".

Test Case V : Expression of Existential Restriction

Existential Restrictions in OWL represent the fact that all the instances of the concept for which an existential restriction is specified upon a property X, must **at least** participate in relationship (via the property X) with instances of a **specific** concept and **may** participate in relationship (via the property X) with instances of any other concept.

We modeled our Fact Set 5.5 to represent the existential restriction over the property "has point of contact" specified for the instances of the concept "Student" in relation to instances of the concept "PersonalContactAddress", as shown below.

**Fact Set 5.5: Expression of Existential Restriction**

Facts:

1. The following classes of Professionals are known:
   a. Student
   b. Professor
2. The following classes of ContactAddress are known:

a. PersonalContactAddress
b. BusinessContactAddress

3. The following are all the known instances of PersonalContactAddress:
   a. HomeAddress
   b. PersonalEmailAddress
4. The following are all the known instances of BusinessContactAddress:
   a. OfficeAddress
   b. CorporateEmailAddress
5. Student always has at least one instance of PersonalContactAddress as their point of contact.
6. In some cases, Student may also have some instances of BusinessContactAddress as their point of contact.

We then asked the question "Describe Student".

Test Case VI : Expression of Disjoints and Aggregation

Here, the aim is to find out how people express the idea that a given concept is logically disjoint from other concepts. In OWL, disjoint concepts are the ones which have no instances in common. Accordingly, we designed our Fact Set 5.6 to present participants with concepts that had none of their instances in common.

Fact Set 5.6: Expression of Disjoints and Aggregation

Facts:

1. The following classes of PopularFootballteam are known:
   a. EnglishTeam
   b. SpanishTeam
   c. ItalianTeam
2. Some known instances of EnglishTeam are Chelsea and ManU.
3. Some known instances of SpanishTeam are RealMadrid and Barcelona.
4. Some known instances of ItalianTeam are A.C. Milan and Juventus.
5. No instance of EnglishTeam is a SpanishTeam and vice versa.
6. No instance of EnglishTeam is an ItalianTeam and vice versa.

We then asked the question "Describe EnglishTeam".

Test Case VII : Expression of Set Operators

Similarly, we designed a test case to identify the preferable pattern of expression for set operations on concepts. For example, below is the test case we designed

for identifying the preferred pattern for expression of the ObjectComplementOf operator

**Fact Set 5.7: Expression of Set Operators (ObjectComplementOf)**

Facts:

1. The following classes of Pizza are known:
   a. NonVegetarianPizza
   b. VegetarianPizza
2. It is known that NonVegetarianPizza implies that the Pizza is not a VegetarianPizza and vice versa.

by posing the question "Describe NonVegetarianPizza" to the participants of the survey.

Test Case VIII : Expression of Siblings

To identify the pattern of expression people prefer in describing the information about sibling concepts of a given concept, we designed a test case that would express the sibling concepts for the concept "Parmense" under its different superconcepts, "MeatyPizza" & "CheeseyPizza" as shown below.

**Fact Set 5.8: Expression of Siblings and Aggregation**

Facts:

1. The following classes of Pizza are known:
   a. MeatyPizza
   b. CheeseyPizza
2. The following classes of MeatyPizza are known:
   a. AmericanHot
   b. LaReine
   c. SloppyGiuseppe
   d. Parmense
   e. Siciliana
   f. PolloAdAstra
3. The following classes of CheeseyPizza are known:
   a. Parmense
   b. Soho
   c. Napoletana
   d. Caprina
   e. LaReine
   f. Rosa
   g. Veneziana

We then asked the question "Describe Parmense. (In particular, how do you express the idea that Parmense is one of the many other kinds under each of its superclasses – MeatyPizza & CheeseyPizza?)".

## 5.2 Survey Procedure

We carried out the survey online, with recruitment from among personal acquaintances, university mailing lists etc. In total, we had 13 participants who responded to all of the test cases.

## 5.3 Survey Results

The responses from participants provide us with varying expressions in natural language text for each of the test cases. Since the responses (from 13 participants) under each test case are varied and subjective, it is only possible for us to categorize the popular pattern of expression from the available responses rather than the popular response itself. Thus, we classify all the available responses under each test cases into separate categories; each category containing responses which follow the same general pattern (with few variations) of expression. In determining the set of responses that fall under the same category, we tried to rely on intuition to come up with the most reasonable generalization that would cover the largest subset of similar responses. Below, we present the distinct categories of response patterns (with a sample response representative of each pattern) observed along with their frequencies under each of the test cases.

Results for Test Case I (Simplification of Concept Name (in Subject Role))

The concept name "GreenPepperTopping" will serve the role of Subject in the output sentence to be generated. With regards to our aim of finding whether or not people prefer to generate the base form and modifier for the concept name in Subject role, we have the following statistics of results:

| S.No. | Pattern (Representative response) | Frequency |
|---|---|---|
| 1 | GreenPepperTopping is a kind of PepperTopping | 8 |
| 2 | Green pepper topping is a kind of pepper topping. | 2 |
| 3 | It is a class of pepper topping. | 3 |

Table 5.1: Results Statistics for Test Case I

From the above results, we observe that most of the participants preferred to keep the concept name in the Subject role (i.e. GreenPepperTopping) intact. Accordingly, we also keep the concept name in the Subject role intact when we generate sentences during realisation.

Results for Test Case II (Simplification of Concept Name (in Object Role), Expression of Named Subsumers and Possibilities of Aggregation)

The concept names ("DeliciousPizza", "CheeseyPizza", "VegetarianPizza" and "RealItalianPizza") will serve the role of Object in the output sentence to be generated. In terms of identifying whether or not people prefer to generate base form and modifier for these concept names (in Object role), we have the following statistics:

| S.No. | Pattern (Representative response) | Frequency |
|---|---|---|
| 1 | Napoletana is a pizza that falls under the class of cheesey, vegetarian, delicious and RealItalian pizza. | 9 |
| 2 | Napoletana is a CheeseyPizza but is also VegetarianPizza, DeliciousPizza and RealItalianPizza. | 4 |

Table 5.2: Results Statistics for Test Case II : Simplification of Concept Name (in Object Role)

Additionally, in terms of identifying the popular mode of expression for named subsumers and their possibility of aggregation in the output sentence, we have the following statistics:

| S.No. | Pattern (Representative response) | Frequency |
|---|---|---|
| 1 | Napoletana is a pizza that falls under the class of Cheesey, Vegetarian, Delicious and RealItalian pizza. | 5 |
| 2 | Napoletana is a pizza that is Cheesey, Vegetarian, Delicious and RealItalian. | 3 |
| 3 | Napoletana is a CheeseyPizza but is also VegetarianPizza, DeliciousPizza and RealItalianPizza. | 2 |
| 4 | Napoletana is a cheesey pizza. Napoletana is a vegetarian pizza. Napoletana is a delicious pizza. Napoletana is a realitalian pizza. | 1 |
| 5 | Napoletana is pizza which has qualities sufficient to place it in any and all known classes of pizza. | 1 |
| 6 | Napoletana is a pizza. | 1 |

Table 5.3: Results Statistics for Test Case II : Expression of Named Subsumers and Possibilities of Aggregation

From the results in table 5.2, we can see that most of the participants preferred to generate the base form and modifier for the concept names in Object role. Similarly, the most popular pattern of expression of named subsumers among the participants was the Pattern 1 in table 5.3. Also, we can see that almost all (10 in total) participants preferred to aggregate all of the available named subsumers while generating sentences. Based on these observations, we check for possibilities to identify base form and modifier for concept names in Object role, adopt the popular pattern to express named subsumers information and aggregate the available named subsumers while generating sentences during realisation.

Results for Test Case III (Expression of Named Equivalents and Aggregation)

From the answers obtained, we have the following statistics:

| S.No. | Pattern (Representative response) | Frequency |
|---|---|---|
| 1 | An instance of WeekDays is also an instance of WorkingDays or BusinessDays. | 5 |
| 2 | WeekDays, WorkingDays and BusinessDays all include the same instances. | 4 |
| 3 | WeekDays, WorkingDays and BusinessDays express the same concept | 2 |
| 4 | WeekDays are the days when one works or makes business. | 1 |
| 5 | BusinessDays, also known as WorkingDays, are WeekDays. | 1 |

Table 5.4: Results Statistics for Test Case III

Based on these observations, we follow the Pattern 1 in table 5.4 to generate text describing and aggregating named equivalents during our realisation phase.

Results for Test Case IV (Expression of Universal Restriction)

The following are the statistics observed:

| S.No. | Pattern (Representative response) | Frequency |
|---|---|---|
| 1 | EUCitizen is a class of citizen that can have free access to EU countries only. However, it might be the case that some instances of EUCitizen don't have any free access rights at all. | 5 |
| 2 | EUCitizen usually have free access to EU countries only. | 4 |
| 3 | EUCitizen is a type of citizen that has free access to EU countries only, unless otherwise specified. | 2 |
| 4 | A few response were irrelevant, like, EUCitizen has free access to Norway. | 2 |

Table 5.5: Results Statistics for Test Case IV

Accordingly, we choose the Pattern 1 in table 5.5 to generate text describing universal restriction axioms during our realisation phase.

Results for Test Case V (Expression of Existential Restriction)

The following are the statistics obtained after categorizing the responses describing the existential restrictions obtained from the survey.

| S.No. | Pattern (Representative response) | Frequency |
|---|---|---|
| 1 | Student is a class of professional that always has at least one personal contact address and may also have other types of point of contact. | 5 |
| 2 | A student always has one personal contact address. Some students can also have other types of point of contact. | 4 |
| 3 | Student is a type of Professionals that have an instance of either PersonalContactAddress or BusinessContactAddress as their point of contact. | 2 |
| 4 | A student can be contacted using PersonalContactAddress or BusinessContactAddress. | 2 |

Table 5.6: Results Statistics for Test Case V

Based on these observations, we choose the most popular pattern, i.e. Pattern 1 from table 5.6, to generate text describing existential restrictions during our realisation phase.

Results for Test Case VI (Expression of Disjoints and Aggregation)

Below, we present the statistics obtained after analyzing the responses collected for the expression and aggregation of disjoint concepts from the survey.

| S.No. | Pattern (Representative response) | Frequency |
|---|---|---|
| 1 | An EnglishTeam is not a SpanishTeam or an ItalianTeam. | 5 |
| 2 | An EnglishTeam cannot be a SpanishTeam or an ItalianTeam. | 2 |
| 3 | EnglishTeam is one of the kinds of popular football team. | 3 |
| 4 | EnglishTeam is a PopularFootBallteam and is a class on it's own. It doesn't fall under Spanish or Italian teams. | 1 |
| 5 | EnglishTeam is a popular football team. | 2 |

Table 5.7: Results Statistics for Test Case VI

From the statistics above, we identify the popular pattern of expression of disjoint concepts (Pattern 1 in table 5.7) and utilize the same in generating descriptions of disjoint axioms during our realisation phase. Also, the statistics reveal that aggregation of the disjoint concepts is desirable while generating our sentences.

Results for Test Case VII (Expression of Set Operators)

The following were the statistics derived from the response collected for the Test Case VII:

| S.No. | Pattern (Representative response) | Frequency |
|---|---|---|
| 1 | A NonVegetarianPizza is a pizza that is not vegetarian. | 6 |
| 2 | NonVegetarianPizza is not a vegetarian pizza. | 4 |
| 3 | NonVegetarianPizza is a type of pizza that is mutually exclusive with vegetarian pizza | 1 |
| 4 | A few responses were irrelevant, such as, NonVegetarianPizza contains ingredients not found in vegetarian pizza. | 2 |

Table 5.8: Results Statistics for Test Case VII

Subsequently, we use the popular pattern (Pattern 1 in table 5.8) during our realisation phase.

Results for Test Case VIII (Expression of Siblings)

Finally, for this category of facts, we have the following statistics:

| S.No. | Pattern (Representative response) | Frequency |
|---|---|---|
| 1 | As a meaty pizza, Parmense is similar to AmericanHot, LaReine, SloppyGuiseppe etc. and as a cheesey pizza, it is similar to Soho, Napoletana, Caprina etc. | 5 |
| 2 | Parmense falls under the class of meaty pizza like AmericanHot, Siciliana etc. and under the class of cheesey pizza like Rosa, Caprina etc. | 2 |
| 3 | Parmense, along with AmericanHot, LaReine, SloppyGuiseppe etc. is a meaty pizza. It is also a cheesy pizza along with Napoletana, Soho, Veneziana etc. | 3 |
| 4 | A few responses were non contextual, like, Parmense is a meaty and cheesey pizza. | 3 |

Table 5.9: Results Statistics for Test Case VIII

We use the popular pattern (Pattern 1 in table 5.9) to express the Siblings information during our realisation phase.

## 5.4 Application of Survey Results

We utilize the most popular pattern of expressions identified from each of the Test Cases to model our realisation task. Equipped with the knowledge of lexical items from the feature structures and identification of most preferred patterns of expression from the survey, we now have sufficient information to set suitable attributes and invoke appropriate methods of the SimpleNLG classes for generating textual output from our system. For example, from the Lexicalisation Proper phase, we have the following feature structure representing the aggregation of stated and implied named subsumers for the concept "Napoletana".

$$\begin{bmatrix} \textbf{Subject} : \text{Napoletana} \\ \textbf{Object} : \text{Pizza} \\ \textbf{ObjectModifier} : [\text{Named, Interesting, Cheesey, RealItalian, NonVegetarian}] \end{bmatrix}$$

In the feature structure, we already have the information about the lexical items that will serve the role of Subject, Object and Object modifier in the output sentence to be generated. From the survey results, we also have the information about the pattern suitable for description of named subsumers. We can now map the lexical items from the feature structure to suit the pattern of expression. Figure 5.1 shows a sample mapping from the feature structure for named subsumers (of the concept "Napoletana") to the pattern identified for the expression of the named subsumers.

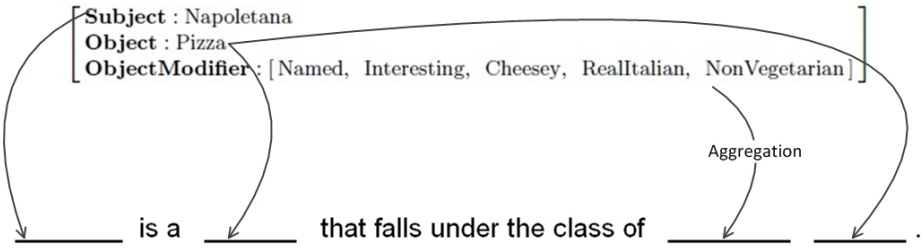


Figure 5.1: Sample mapping from Feature Structure to Expression Pattern

As discussed in section 4.5, generating this sentence within our programming environment is carried out by calling appropriate methods of the SimpleNLG realisation module. Similarly, the realisation of other set of feature structures from the Lexicalisation Proper phase is guided by the corresponding patterns of expression identified from the survey. In effect, we now have a complete modular approach to generation of text from our feature structures.

In the following section, we present few sample outputs – sentences generated from our system to respond to the variety of factoid questions. As per the discussion in section 4.3.1, the answer to "What is $X$?" question is generated by processing all categories of axioms pertaining to the concept $X$ (effectively, the full document plan tree generated for the concept $X$) and the answer to "How to identify a $X$?" question is generated by processing the category of equivalent axioms (both stated and implied, when present) pertaining to the concept $X$ (effectively, the "Equivalents" portion of document plan tree generated for the concept $X$). Finally, the answer to "What are the varieties of $X$?" question is generated by the listing the named subconcepts (stated and implied) of the concept $X$.

## 5.5 Sample Results

**Question:** What is IceCream?

**Answer:** IceCream falls under the class of food. As a food, IceCream is similar to Pizza, PizzaTopping and PizzaBase. IceCream always has at least one fruit topping and may also have other types of toppings. An IceCream is not a Pizza or a PizzaBase or a PizzaTopping.

**Question:** What is VegetarianPizzaEquivalent2?

**Answer:** VegetarianPizzaEquivalent2 is a pizza that falls under the class of vegetarian pizza. An instance of VegetarianPizzaEquivalent2 is also an instance of VegetarianPizzaEquivalent1. VegetarianPizzaEquivalent2 is a class of pizza that can have toppings of cheese, fruit, herbspice, nut, sauce and vegetable only. However, it might be the case that some instances of VegetarianPizzaEquivalent2 don't have any toppings at all.

**Question:** What is RealItalianPizza?

**Answer:** RealItalianPizza is a pizza that falls under the class of thinandcrispy pizza. RealItalianPizza can have base of thinandcrispy only. However, it might be the case that some instances of RealItalianPizza don't have any base at all. RealItalianPizza has Italy as it's countryoforigin.

**Question:** What is SpicyPizzaEquivalent?

**Answer:** SpicyPizzaEquivalent falls under the class of pizza. As a pizza, SpicyPizzaEquivalent is similar to NamedPizza, NonVegetarianPizza, InterestingPizza etc. An instance of SpicyPizzaEquivalent is also an instance of spicy pizza. SpicyPizzaEquivalent is a class of pizza that always has at least one pizza topping of hot spice and may also have other types of toppings.

**Question:** What is Rosa?

**Answer:** Rosa is a pizza that falls under the class of named, VegetarianPizzaEquivalent1/VegetarianPizzaEquivalent2, interesting and cheesey pizza. As a named pizza, Rosa is similar to SloppyGiuseppe, FourSeasons, UnclosedPizza etc., as a VegetarianPizzaEquivalent1/VegetarianPizzaEquivalent2, it is similar to Soho, PrinceCarlo, Giardiniera etc., as an interesting pizza, it is similar to Mushroom, AmericanHot, LaReine etc. and as a cheesey pizza, it is similar to Capricciosa, Veneziana, Parmense etc.

Rosa always has at least one mozzarella with gorgonzola and tomato topping and may also have other types of toppings. Rosa can have toppings of gorgonzola, mozzarella and tomato only. However, it might be the case that some instances of Rosa don't have any toppings at all. A Rosa is not a Giardiniera or a Capricciosa or a PrinceCarlo or a QuattroFormaggi or a LaReine or a Veneziana or a Napoletana or a Mushroom or a SloppyGiuseppe or a Caprina or a UnclosedPizza or a American or a Fiorentina or a Parmense or a FruttiDiMare or a PolloAdAstra or a FourSeasons or a AmericanHot or a Soho or a Siciliana or a Cajun or a Margherita.

**Question:** How to identify a MeatyPizza?

**Answer:** Any pizza that has at least one meat topping is a MeatyPizza.

**Question:** How to identify a RealItalianPizza?

**Answer:** Any pizza that has Italy as it's countryoforigin is a RealItalianPizza.

**Question:** How to identify an InterestingPizza?

**Answer:** Any pizza that has a minimum of 3 different toppings is an InterestingPizza.

**Question:** How to identify a NonVegetarianPizza?

**Answer:** Any pizza that is not VegetarianPizza is a NonVegetarianPizza.

**Question:** What are the varieties of RealItalianPizza?

**Answer:** Following are the all known varieties of RealItalianPizza:

- Napoletana
- Veneziana

**Question:** What are the varieties of SpicyTopping?

**Answer:** Following are the all known varieties of SpicyTopping:

- CajunSpiceTopping
- HotGreenPepperTopping
- HotSpicedBeefTopping
- JalepenoPepperTopping
- TobascoPepperSauce

**Question:** What are the varieties of IceCream?

**Answer:** There are no known varieties of IceCream.

## 5.6 Discussion

In terms of identifying the lexical items for sentence generation, we have entirely based our work on exploiting the structural patterns common in naming of concepts and relations during ontology authoring. When such attempts have been feasible, we have obtained grammatical lexical items, as in the sentence "Rosa is a **pizza** that falls under the class of **named pizza**". However, the structural similarities can not be taken for granted and attempts relying purely on such techniques will often fail; producing lexical items that are not really words of English, as in the sentence "RealItalianPizza has Italy as it's **countryoforigin**". In our work, we have accepted lexicalisations resulting in both valid and invalid words of English as lexical items; our goal here is to generate sentences that respond to a particular question by conveying the semantics behind the facts expressed by various axioms associated to concepts in ontologies, which we believe that sentences like "Rosa is a pizza that falls under the class of named pizza" and "RealItalianPizza has Italy as it's countryoforigin", both, truly achieve. In existing works, authors have targeted generation of true words of English as lexical items from the concept and relation names in ontologies in various ways.

Mellish & Sun [28] rely on separators such as capital letters (as in CheeseTopping), underscores (as in red_wine) etc. to breakup a complex concept name, such as "CheeseTopping" into lexicons "Cheese" and "Topping". Androutsopoulos & Galanis [14] go further in suggesting that an external annotation of OWL ontologies to associate the ontological resources (concept and relation names) with domain-dependent linguistic resources (lexical items, templates etc.) is necessary for natural language generation from ontologies. However, we have also observed, particularly with regards to the question-answer scenario, that people do not always prefer to break up a complex concept name into simpler lexical items (as in Test Case I, where participants preferred to keep the name for the concept in subjective role intact). It would be interesting to investigate how appropriately lexicalisation could be carried out to address such needs.

# Chapter 6

# Conclusion

We have presented a generic approach in Natural Language Generation from ontologies. We have identified factoid questions that can be posed upon the knowledge base in ontologies and discussed with detailed and structured methods of transiting from the logical representation of knowledge in ontologies to their textual descriptions in natural language. In particular, we have put efforts in describing the class axioms present in ontologies; we have identified, classified and processed several varieties of those axioms in coming up with our own plan to generate natural language text. We have identified the categories and order of axioms, beforehand, that any NLG developer can pursue to come up with similar generation systems. In addition to the explicitly stated knowledge available in the axioms, we have utilized the services of a reasoner in extracting implicit knowledge from the ontology and identified additional constructs (such as the Siblings category of axioms we custom defined in section 4.2) to broaden up the "content" for our system. Further, we have identified the category of axioms that best serve to generate a response to each type of the factoid questions respectively. In this way, we believe that we have significantly addressed the content determination task for similar systems. We have discussed and implemented generic procedures of identifying lexical items and their roles. The factoid questions we have determined are effectively a stipulation of the communicative goals of our system and have overall shaped our generation system; from the determination of content to the empirical study carried out for making informed decisions on identifying the best pattern of expression of the ontological facts into natural language text. Thus, from a theoretical perspective, one can view the present work as bridging between QA (as traditionally understood) and NLG.

While we have used the pizza ontology as a reference for presenting the procedure, the methods we have designed in carrying out those procedures is generic enough to be applicable to other ontologies as well. The pizza ontology already provides numerous varieties of axioms that are likely to be encountered in

other ontologies; thus we have sufficiently addressed our procedure for a broad set of ontologies. We have also discussed possible implementation model for variations not encountered within the pizza ontology (for example, the aggregation of stated subsumers with Universal restrictions and the aggregation of stated subsumers with HasValue restrictions in section 4.4.2); we believe that it should not be very difficult for anyone to formulate similar implementation model (resembling our methodology) to address newer varieties.

## 6.1 Application Context

We can envisage numerous application scenarios for our system. Given that the framework described is designed to provide answers to questions posed over concept names in the ontology, there are at least three immediate applications:

- Empower understanding of ontological facts to a beginner.
- Serve an ontology author in suitably expressing the knowledge he/she has authored into his/her ontologies to general people.
- Facilitate an expert in reviewing the facts expressed by an author in his/her ontology

Particularly, with large scale ontologies being popular to represent complex chains of relationships among several participating concepts in a huge domain such as the biomedical domain; during the course of design, operation and maintenance, a human author/expert/user would perhaps like to comprehend the current state of knowledge within the ontology in terms of intuitive natural language text rather than by exploring the ontology tree hierarchy and rebuilding a mental model of the knowledge base each time. This is where we presume our system comes in handy. From a long-term perspective, finding general ways of generating texts from ontologies is a scientifically useful tool: since an ontology is a general description of some domain, the application of NLG techniques and their validation, can serve to help us in understanding how we talk about specific domains of knowledge.

## 6.2 Future Work

Apart from the issue of generating proper lexical items, which we discussed in section 5.6, we see other possibilities to improve upon the results of the current system. One possibility is in terms of realising "closure restrictions" – a condition in which a universal restriction specified over a property has a filler that is the logical union of the fillers specified for existential restrictions over the same property. As an example from the pizza ontology itself, we have the following universal and existential restriction axioms specified for the concept "Rosa":

SubClassOf(<Rosa> ObjectAllValuesFrom (<hasTopping>
ObjectUnionOf (<GorgonzolaTopping> <MozzarellaTopping> <TomatoTopping>)))

SubClassOf(<Rosa> ObjectSomeValuesFrom(<hasTopping> <GorgonzolaTopping>))

SubClassOf(<Rosa> ObjectSomeValuesFrom(<hasTopping> <MozzarellaTopping>))

SubClassOf(<Rosa> ObjectSomeValuesFrom(<hasTopping> <TomatoTopping>))

As we can see, the axiom expressing the universal restriction has the filler that is the logical union (ObjectUnionOf) of the fillers specified for the remaining three separate existential restriction axioms over the same property "hasTopping", thus fulfilling the criteria for "closure restriction". Semantically, when such a "closure restriction" is present, it eliminates the possibility of the concept (Rosa) not participating in any "hasTopping" relation (the "none" part of the universal restriction axioms, as we discussed for Test Case IV in section 5.1). It also eliminates the possibility of participating via the "hasTopping" relation with fillers other than those specified – GorgonzolaTopping, MozzarellaTopping and TomatoTopping (the open world assumption which is default with existential restrictions, as presented with an example in Test Case V in section 5.1). In turn, the realisation obtained for this set of universal and existential restrictions could be fairly simpler, perhaps along the lines of "Rosa is a class of pizza that only has mozzarella with gorgonzola and tomato topping" instead of the verbose statements "Rosa always has at least one mozzarella with gorgonzola and tomato topping and may also have other types of toppings. Rosa can have toppings of gorgonzola, mozzarella and tomato only. However, it might be the case that some instances of Rosa don't have any toppings at all." that we have presented in section 5.5.

In existing work, we haven't targeted survey questionnaire with similar situation of closure restrictions; rather we used the popular pattern of expression of universal and existential restrictions identified from the survey, in turn, to generate the sentences describing universal and existential restrictions of the closure restrictions respectively; thereby generating the verbose statements. While it should not be very difficult to compute feature structures for representing closure restriction axioms – either by deploying a theorem prover to derive a better OWL statement uniting both the restrictions and then generating a feature structure based on the new statement or by provisioning a suitable aggregation strategy, similar to various cases we discussed in section 4.4.2, between the feature structure representing universal restriction and the feature structure representing existential restriction of the closure restriction – from their respective universal and existential restriction counterparts, we have yet to identify the popular pattern of expression of such closure restrictions. In future, an additional test case could be developed by providing the participants with both the

universal and existential restrictions acting over the same concept and asking them to come up with a response. The responses so obtained could then be used to designate our pattern of expression of closure restrictions.

With regards to the factoid question "How to identify a $X$?", we have looked into ways of generating answers by using the stated equivalence axioms for the given concept name $X$ in the ontology. This approach could be scaled up by taking into account the set of properties present in the ontology, which when combined (via some logical operations such as the complement, intersection and quantifiers etc.), are sufficient to distinguish the concept $X$ from others in the ontology. For example, Ren et al. [39] present and discuss their Description Logic based approach in generating OWL expression which represents a singleton set for a given individual in the ontology; thereby generating a referring expression which uniquely identifies that individual in the ontology. Similar approach would enable us in generating answers to the "How to identify a $X$?" questions in terms of the conjugating properties that uniquely identify the concept $X$ rather than relying exclusively on the presence of equivalence axioms stated for the given concept $X$ in the ontology.

As of now, our system is not capable of processing inherited class restrictions. For example, for the concepts "Rosa" and "RealItalianPizza", among others in the pizza ontology, the existential restriction specified over the relation "hasBase" gets inherited from their superconcept "Pizza", as shown below:

SubClassOf(<Rosa> ObjectSomeValuesFrom(<hasBase> <PizzaBase>))

SubClassOf(<RealItalianPizza> ObjectSomeValuesFrom(<hasBase> <PizzaBase>))

While in some cases (for example, the answer generated for the question "What is Rosa?" in section 5.5) this incapability simply means an omission of information, in other cases (for example, the answer generated for the question "What is RealItalianPizza?" in section 5.5), it would lead to misleading information – we have generated text defining "RealItalianPizza" as "RealItalianPizza can have base of thinandcrispy only. However, it might be the case that some instances of RealItalianPizza don't have any base at all."; however this is not completely true because the inherited restriction states that all instances of "RealItalianPizza" must have at least one base (of type "PizzaBase") and thus rules out the possibility for instances of "RealItalianPizza" not having any base. How such restrictions could be retrieved, represented and processed is also an interesting question for future work. In particular, such complex semantic inferences open up many possibilities for aggregation and realisation, and the best choice among these is an interesting empirical question in its own right.

We have yet to see how our approach could be extended in terms of generating natural language text from ontologies that involve a merging of one or more other ontologies within them and ontologies that are augmented with rule markup

languages, such as SWRL (Semantic Web Rule Language). It would also be interesting to extend our work in terms of describing individuals and relations present in ontologies.

All of these are essentially instances of a more general problem, namely: how to communicate logically complex information in natural language, allowing for complex patterns of inference. Investigating such questions in depth is bound to yield more insight into the connection between abstract, formal knowledge representation and reasoning and human communication. This is an area in which NLG – as a concrete way of making machines communicate in a more humanlike manner – can contribute to a broader understanding of how linguistic communication interacts with the vast repositories of knowledge that humans have at their disposal.

# Bibliography


[1] Albert Gatt A, François Portet, Ehud Reiter, James Hunter, Saad Mahamood, Wendy Moncur, and Somayajulu Sripada. From data to text in the neonatal intensive care unit: Using NLG technology for decision support and information management. *AI Communications*, 22(3):153–186, 2009.

[2] Giovanni Adorni and Michael Zock, editors. *Trends in Natural Language Generation, An Artificial Intelligence Perspective, Fourth European Workshop, EWNLG '93, pages 1–15*, volume 1036 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[3] Tej Anand and Gary Kahn. Making sense of gigabytes: a system for knowledge-based market analysis. In *Proceedings of the fourth conference on Innovative applications of artificial intelligence*, IAAI'92, pages 57–69. AAAI Press, 1992.

[4] Lora Aroyo, Pieter Bellekens, Martin Björkman, Jeen Broekstra, and Geert-Jan Houben. Ontology-based personalization in user-adaptive systems. In *2nd International Workshop on Web Personalization, Recommender Systems and Intelligent User Interfaces (WPRSIUI'06)*, 2006.

[5] Fredrik Arvidsson and Annika Flycht-Eriksson. Ontologies I. `http://www.ida.liu.se/~janma/SemWeb/Slides/ontologies1.pdf`.

[6] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. Owl Web Ontology Language Reference. `http://www.w3.org/TR/owl-ref/`, 2004.

[7] Thomas Bethem, Janet Burton, Ted Caldwell, Mike Evans, Richard Kittredge, Benoit Lavoie, and Jennifer Werner. Generation of real-time narrative summaries for real-time water levels and meteorological observations in ports®. In *Proceedings of the Fourth Conference on Artificial Intelligence Applications to Environmental Sciences*, AMS-2005, 2005.

[8] Kalina Bontcheva. Generating tailored textual summaries from ontologies. In *ESWC*, volume 3532 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2005.

[9] Dana Dannélls. Discourse generation from formal specifications using the grammatical framework, GF. 2010.

[10] Claudia Denicia-Carral. Respondiendo Preguntas de Definición mediante el Descubrimiento de Patrones Léxicos. Master's thesis, Instituto Nacional de Astrofísica, Óptica y Electrónica, Tonantzintla, Puebla, 2007.

[11] Nick Drummond, Matthew Horridge, Robert Stevens, Chris Wroe, and Sandra Sampaio. Pizza ontology. `http://www.co-ode.org/ontologies/pizza/2007/02/12/`, 2007.

[12] Marc Ehrig and York Sure. Ontology mapping - an integrated approach. In *ESWS'04*, pages 76–91. Springer Verlag, 2004.

[13] Mark S. Fox. The tove project towards a common-sense model of the enterprise. In *Proceedings of the 5th international conference on Industrial and engineering applications of artificial intelligence and expert systems*, IEA/AIE '92, pages 25–34, London, UK, 1992. Springer-Verlag.

[14] Dimitrios Galanis and Ion Androutsopoulos. Generating multilingual descriptions from linguistically annotated owl ontologies: the naturalowl system. In *In Proceedings of the 11th European Workshop on Natural Language Generation, Schloss Dagstuhl*, 2007.

[15] Albert Gatt and Ehud Reiter. SimpleNLG: A realisation engine for practical applications. In *Proceedings of the 12th European Workshop on Natural Language Generation*, ENLG '09, pages 90–93, 2009.

[16] Gene Ontology Consortium. The Gene Ontology GO database and informatics resource. *Nucleic acids research*, 32(1):D258–D261, 2004.

[17] Eli Goldberg, Norbert Driedger, and Richard I. Kittredge. Using natural-language processing to produce weather forecasts. *IEEE Expert: Intelligent Systems and Their Applications*, 9:45–53, April 1994.

[18] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, June 1993.

[19] Jerry R. Hobbs and Robert C. Moore, editors. *Formal Theories of the Commonsense World*. Greenwood Publishing Group Inc., Westport, CT, USA, 1985.

[20] Petr Knoth, Trevor Collins, Elsa Sklavounou, and Zdenek Zdrahal. Facilitating cross-language retrieval and machine translation by multilingual domain ontologies. In *Workshop on Supporting eLearning with Language Resources and Semantic Data (at LREC 2010)*, May 2010.

[21] Sven J. Körner and Torben Brumm. Natural language specification improvement with ontologies. *International Journal of Semantic Computing (IJSC)*, 03(04):445–470, 2010.

[22] Yen-Ting Kuo, Andrew Lonie, Liz Sonenberg, and Kathy Paizis. Domain ontology driven data mining: a medical case study. In *Proceedings of the 2007 international workshop on Domain driven data mining*, DDDM '07, pages 11–17, New York, NY, USA, 2007. ACM.

[23] François Mairesse and Marilyn Walker. Personage: Personality generation for dialogue. In *In Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL*, pages 496–503, 2007.

[24] W. C. Mann and S. A. Thompson. Rhetorical structure theory: toward a functional theory of text organization. *Text*, 3:243–281, 1998.

[25] J.H. Martin and D. Jurafsky. *Speech and Language Processing.* Prentice Hall, 2008.

[26] C. Mellish and X. Sun. Natural Language Directed Inference in the Presentation of Ontologies. In *Procs of the 10th European Workshop on Natural Language Generation, Aberdeen, 2005*, Aberdeen, UK, 2005.

[27] Chris Mellish and Jeff Z. Pan. Natural language directed inference from ontologies. *Artificial Intelligence*, 172(10):1285 – 1315, 2008.

[28] Chris Mellish and Xiantang Sun. The semantic web as a linguistic resource: opportunities for natural language generation. knowledge based systems. In *Presented at the Twenty-sixth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, 2005.

[29] Marvin Minsky. A framework for representing knowledge. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1974.

[30] Riichiro Mizoguchi and Mitsuru Ikeda. Towards ontology engineering. Technical report, I.S.I.R., Osaka University, 1997.

[31] Alexander Osterwalder and Yves Pigneur. An e-business model ontology for modeling e-business. Technical Report 0202004, EconWPA, February 2002.

[32] Jeff Z. Pan and Chris Mellish. Supporting semi-automatic semantic annotation of multimedia resources. In *in: Proceedings of the 3rd IFIP Conference on Artificial Intelligence Applications & Innovations (AIAI 2006*, pages 609–617. Sage Publications, 1996.

[33] Richard Power. Towards a generation-based semantic web authoring tool. In *Proceedings of the 12th European Workshop on Natural Language Generation*, ENLG '09, pages 9–15, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[34] Aarne Ranta. Grammatical Framework: A Type-Theoretical Grammar Formalism. *Journal of Functional Programming*, 14(02):145–189, March 2004.

[35] Ehud Reiter. An architecture for data-to-text systems. In *Proceedings of the Eleventh European Workshop on Natural Language Generation*, ENLG '07, pages 97–104, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.

[36] Ehud Reiter. *The Handbook of Computational Linguistics and Natural Language Processing*, chapter Natural Language Generation. Blackwell Publishing, 2010.

[37] Ehud Reiter and Robert Dale. *Building natural language generation systems*. Cambridge University Press, New York, NY, USA, 2000.

[38] Ehud Reiter, Somayajulu Sripada, Jim Hunter, and Ian Davy. Choosing words in computer-generated weather forecasts. *Artificial Intelligence*, 167:137–169, 2005.

[39] Yuan Ren, Kees van Deemter, and Jeff Z. Pan. Charting the potential of description logic for the generation of referring expressions. In *Proceedings of the 6th International Natural Language Generation Conference*, INLG '10, pages 115–123, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[40] J. F. Sowa. *Conceptual structures: information processing in mind and machine*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[41] Xiantang Sun and Chris Mellish. Domain independent sentence generation from rdf representations for the semantic web. In *Presented at the ECAI06 Combined Workshop on Language-Enabled Educational Technology and Development and Evaluation of Robust Spoken Dialogue Systems*, 2006.

[42] Kees Van Deemter, Emiel Krahmer, and Mariët Theune. Real versus template-based natural language generation: A false opposition? *Computational Linguistics*, 31:15–24, March 2005.

[43] Scott Farrar William, William D. Lewis, and D. Terence Langendoen. A common ontology for linguistic concepts. In *In Proceedings of the Knowledge Technologies Conference*, pages 10–13, 2002.